Contents

Introduction

Connecting to a Server
Joining a Channel
Chatting Privately

Sending & Receiving files
Changing Colors
Setting Options

Basic IRC Commands
mIRC Commands

Aliases, Popups, & Remote
Variables & Identifiers

Other Features

The Author
License Agreement
How to Register

Acknowledgements

Introduction

Welcome to **mIRC**, an Internet Relay Chat Client.

mIRC attempts to provide a user-friendly interface for use with the **Internet Relay Chat** network. The IRC network is a **virtual meeting place** where people from all over the world can meet and talk.

To IRC all you need to do is Connect to a server, Join a channel, and Chat!

mIRC will guide you through these initial stages and hopefully you'll be chatting in no time. If you get stuck or want to find out more about a certain feature, just click on a Help button eg. in the Options dialog, and you should find some hints there to help you.

As you become more **experienced** you can also start **configuring** mIRC's features to suit your own needs and tastes, features such as colors, fonts, function keys, aliases, popup menus, scripts, sounds and many others.

The first **few** sections of this help file are **highly** recommended reading as they provide a quick introduction to all of the **basic** aspects of IRC, which will help you get started.

Please also remember to visit the **mIRC website** at http://www.mirc.com regularly for the latest information on mIRC.

Joining A Channel

Once you've connected to an IRC Server, you can join a channel to talk to other people. There are several ways to join a channel, each is explained below.

The Channels Folder
The easiest way to join a channel is to use the channels folder which holds a list of your favourite channels. mIRC **automatically** pops up this folder the moment you connect to an IRC Server. You can join one of the listed channels by selecting it and clicking the **Join** button.

You can also view the channels folder by clicking on the Channels Folder button in the toolbar.

The Channels List
Another way to join a channel is to retrieve the list of currently active channels by using the List Channels dialog. You can view the List Channels dialog by clicking on the List Channels button in the toolbar.

To retrieve the **entire** channels list, you can click on the **Get List** button. The list can be quite long, often thousands of channels, so it can take several minutes to retrieve it. mIRC will save this list once you have retrieved it, so if you wish to view it again later you can just click on the **Apply** button. However, if you want an updated list you will need to retrieve it again.

If you click your right mouse button in the channels list window, a popup menu with various options will appear.

Note: You can also specify a filename for the channels list which can be useful if you regularly visit different IRC networks.

The /join Command
The format of the /join command, which is an IRC Command, is **/join #channel** where #channel is the name of the channel you want to join. So if you wanted to join channel #mIRC, you would type **/join #mIRC** and press enter, and a moment later the #mIRC window will open indicating that you have joined it.

Creating a new channel
You can create a new channel if it doesn't already exist just by joining it. So, let's say you want to create a channel called #bubbles, you would just type **/join #bubbles**, and if it doesn't exist it will be created for you. If it does exist, you will just join it as usual.

Talking on a channel
You can talk to other people by typing in a message and pressing the enter key. Your message will be sent to the channel and everyone on the channel will see it. A good first message is just to say hello with a smiley face :)

The **listbox** on the side of the channel window lists all of the people who are currently on that channel. If you click your right mouse button in the listbox, a popup menu with various options will appear.

Popup menus are actually used everywhere in mIRC, you can even click your right mouse button in the status window, or in the channel window itself, and a different popup menu will appear. These popup menus are configurable, you can change them in the Popups dialog to perform whatever functions you require.

Leaving a channel
You can leave a channel by clicking it's window's close button, or you can use the **/part** command, which is another IRC command similar to /join. The format of the /part command is **/part #channel** where #channel is the name of the channel you want to leave. If you type **/part** without a channel name and press enter, you will part the current channel.

Hint: you can click the top left corner button/icon in any window in mIRC to view the System menu which contains useful features.

Chatting Privately

As well as being able to chat on public **channels**, mIRC also allows you to chat **privately** with someone.

If you're **on a channel**, and you see someone you'd like to chat with, you can **double-click** on their nickname in the nickname listbox and a private query window will open up. You can then start chatting privately to them through the query window. Alternatively, you can **click your right mouse button** in the nickname listbox and a **popup menu** will appear with various options, one of which will be to open a private query window to the person selected in the nickname listbox.

If you're **not on a channel**, you can type the command **/query nickname**, where nickname is the person you want to chat with. Press the enter key, and a query window will open up and you can start chatting privately, assuming of course that the person is on IRC. You can **find out** if a person is on IRC by using the **/whois nickname** command.

There is another way to chat privately called **DCC Chat**. This method is more secure and usually faster because it doesn't rely on the IRC Server to relay your messages. Instead it connects **directly** to the other person's IRC Client. However it does need to use the IRC Server to **initiate** the chat session.

To DCC Chat someone, you can click on the **Chat button** in the toolbar, and a DCC Chat dialog will pop up. Enter the person's nickname, and click on the Chat button, and if the person **accepts** your DCC Chat request, you will be able to start talking to them privately.

If a user sends **you** a chat request, a chat dialog will pop up asking you whether you want to accept their chat request. You can then accept or decline. You can find out more about DCC Chat related settings in the DCC Section.

The **/dcc chat <nickname>** command is another way of initiating a dcc chat, where nickname is the user you with whom want to dcc chat.

Note: DCC Chat needs to use your IP address to initiate a connection with another client. If you are having trouble initiating a connection then your IP address might be wrong. See the Local Info section for more information.

Sending & Receiving Files

The ability to Send and Receive files is one of the most useful features of mIRC since it allows you to share all kinds of information with other people on IRC.

Warning: If you've never shared files before, please read the Accepting Files on IRC section so as to be aware of the **dangers** of accepting files from others before you start.

On IRC, a method called DCC Send and DCC Get is used to connect **directly** to another IRC client to **Send** and **Get** files, instead of going through the IRC network. The IRC network is used only to initiate the DCC Send request.

DCC Send

DCC Send allows you to send a file to another user. You can do this by clicking on the DCC Send toolbar button to open the DCC Send dialog. You can then enter the nickname of the user, select the file you want to send, and click on the Send button.

mIRC will then tell the user that you want to send them a file. The user then has to accept your send request, at which point the file transfer will begin.

Packet Size

The packet size is the number of bytes that mIRC will send to another client in one packet. The minimum is 512, the maximum is 8192.

If you check **minimize** then the dcc send window(s) will be minimized automatically.

Fill Spaces

The fill spaces option is only available in the 32bit version and only under operating systems that allow spaces in filenames. It is highly recommended that you leave this option turned on. For more information read this.

Fast Send

Turning on this option should speed up your DCC sessions. You can also change this setting with the **/fsend [on|off]** command.

Note: DCC Send needs to use your IP address to initiate a connection with another client. If you are having trouble initiating a connection then your IP address might be wrong. See the Local Info section for more information.

DCC Get

Whenever someone tries to DCC Send a file to you, mIRC pops up the DCC Get and asks you if you want to accept the file. If you choose to **accept** the file, mIRC will ask the sender to begin the file transfer, at which point you should begin receiving the file.

DCC Resume

This feature allows you to resume DCC transfers that failed to complete.

If a user tries to send you a file that already exists in your get directory then you will be shown a warning that the file exists. You then have the option to either overwrite, resume, or rename the file.

If you select overwrite then the whole file will be downloaded from the beginning and any existing file of the same name will be erased.

If you select resume then mIRC will attempt to negotiate a transfer resume to get the remaining part of the file. It will append this to the portion of the file you already have.

The mIRC DCC Resume Protocol is described here if you need to know how it works.

DCC Options

You can also find other DCC related settings in the DCC dialog which can affect how file transfers behave.

The /dcc send command

The /dcc send command can also be used to initiate a DCC Send to the specified nickname. The format of the command is:

/dcc send -clN <nick> <file1> [file2] ... [fileN]

If you specify more than one filename, multiple dcc send sessions to the specified user are initiated.

If you specify a wildcard filename, then the DCC Send dialog will display files matching the wildcard.

The **-c** switch makes the dcc window close automatically once the transfer has finished.

The **-lN** switch limits the transfer rate to N Kb/s, see the DCC section for more information.

Note: If you want to connect to someone else's DCC Server you can specify an IP Address and port instead of the nickname, eg. /dcc send ipaddress:port <file1> ...

Changing Colors

mIRC uses **black text** on a **white background** by default because this allows **colored** text to appear clearly and crisply. However, if you find the color scheme too bright, or if you want something a bit more fun, you can change the text and background, as well as other specific types of text eg. your own messages, to other colors.

To change the color settings, select the **Colors dialog** from the **Tools Menu**. It will popup up and show you the different types of messages in their current color settings.

To **change** the color of an **item**, just **click** on it and you'll see it's **name** appear in the **listbox** at the bottom. Then **click** on any of the **colored squares**, and you'll see the color of the text for that item change accordingly.

To **change** the **background color**, just **click** on the background itself and select a color as usual. You can also click on the background of the **editbox** and the **listbox** to change their colors as well.

If you don't like the changes you've made you can either click **cancel** or you can click the **reset** button which will reset the colors back to the **default** mIRC colors.

To **customize** the color of one of the color boxes, just click your **right mouse button** on it and a custom color dialog will pop up allowing you to choose a new color for it.

Once you've **finished** making changes, click on the **Ok** button, and hey presto you should now feel a bit more colorful!

mIRC also allows you to **interactively** change the color, as well as the **appearance**, of text as you type. You can find out how to do this in the Control Codes section. However remember that other people might have a background color that's **different** from yours, so what appears clearly on your background color might be hard to read on someone else's.

Setting Options

The Options dialog allows you to change most of the settings which determine how mIRC works for you.

- Connect
  - Options
  - Local Info
  - Identd
  - Firewall
- IRC
  - Options
  - Perform
  - Highlight
  - Messages
  - Catcher
  - Logging
  - Flood
- Sounds
  - Requests
  - Agents
- Mouse
  - Drag Drop
- DCC
  - Options
- Display
  - Options
  - Windows
  - Tray
- General
  - DDE Server
  - Finger
- Server
  - Lock

Connecting to a Server

Connecting to an IRC server is the first step to chatting on IRC and is done through the **Connect** dialog, which pops up automatically when you first run mIRC.

Once you enter a few **basic** pieces of information about yourself, you can select an **IRC Server** from the servers list and then click on the **Connect button**. You'll know you've connected when the IRC server shows you it's **Message Of The Day**, which contains information about that server. At that point you'll be able to Join a channel to start chatting.

Basic information

The following information is required before you can connect to a server.

Full Name
You can enter your real name here, however note that whatever you enter can be seen by other people on IRC. Most people usually enter a witty one-liner or comment.

Email Address
You must enter a full email address eg. khaled@mirc.com.

Nickname and Alternative
Your nickname is the name by which people will know you on IRC. Remember that there are many hundreds of thousands of people on IRC, so it's possible that someone might already be using the nickname you've chosen. If that's the case, you should try to pick a different, more unique, nickname. You can enter an alternative nickname as well in case someone is using your first nickname. If both nicknames are in use, mIRC inserts "/nick" into the edit box so that all you have to do is enter a new nickname and press enter.

Invisible Mode
If you turn on the invisible mode switch, people will not be able to find you on IRC unless they already know your nickname, or if you join a channel or talk to them privately.

IRC Servers

The IRC server that you choose is the most important factor in determining how quickly and easily you connect, so if it's taking a long time to connect to one IRC Server, choose a different one and try connecting again.

Note: If you're having problems connecting to an IRC Server, see the Connection Problems section.

You can manage your list of IRC servers by using the add, edit, and delete buttons. Each IRC server consists of the following information:

Description
This can be any text you want and serves only as a description.

Address
This is the IRC server address eg. irc.dal.net

Port Number
This is usually 6667. If the server allows connections on different ports, you can enter them all separated by commas eg. 6667,6668,6669 and mIRC will pick one randomly each time it connects to the server.

Group Name
This allows you to group servers together when they are sorted with the **Sort** button.

Password
This is **rarely** required, so you should not have to enter anything here unless you have been specifically told to do so.

Hint: If you click your right mouse button on the Options toolbar button, a popup menu appears that allows you to quickly connect to an IRC server.

New Server Window

The **new server window** option allows you to connect to more than one IRC server at the same time. Just check the new server window checkbox, and click the connect button. The checkbox setting isn't remembered, so it will be **turned off** the next time you open the connect dialog.

To open a new server window **without** connecting to a server, you can check the new server window checkbox and then press the OK button.

Hint: You can click your **right mouse button** on the options toolbar button to display a popup menu listing your most **recently** accessed IRC servers. If you press the **shift key** while selecting a server in the popup menu it will open in a new server window.

Connect Options

Connect on startup
Makes mIRC connect to the default IRC server automatically when it is run.

Reconnect on disconnection
Makes mIRC reconnect automatically to a server if you were disconnected and you didn't type /quit.

Popup connect dialog on startup
Pops up the connect dialog when you run mIRC.

Move to top of list on connect
Moves a server to the top of the servers list when you connect to it.

Default Port
The default server port that will be used when connecting if one wasn't specified for that server.

Retry...
Retries connecting to a server the specified number of times if the connection attempt failed.

Advanced...
The **random local ports** option makes mIRC use non-consecutive ports when creating sockets, this might help against DoS attacks trying to disable your server or dcc connections.

The **bind sockets to IP** option makes mIRC bind all sockets to a specific local IP address, in case you want it to use a different outgoing network connection.

Multi-Server...
Allows you to change various multi-server related settings.

Local Info

Your **Local Host** and **IP Address** are needed if you want to use the DCC capabilities of mIRC.

mIRC will try to look up these values by itself and display them in the **Setup** dialog. However, if you see the message **Unable to resolve Local host** or you are unable to **initiate DCC sessions**, then changing the settings below might help.

If you see the message **Unable to resolve Local host** with the **32bit version of mIRC**, the problem might be related to using a **16bit winsock**, so you should try out the **16bit version of mIRC** to see if it works for you.

Local Host
This is used to register with the server and may be the part of your email address after the @ sign, eg. if my email address is khaled@mirc.com, then I would enter mirc.com here.

If you leave this box empty then mIRC will try to get your local hostname by itself. However, if mIRC replies with the message **Unable to get local hostname** then you will have to fill in your local hostname manually. mIRC will then use whatever you have entered to get your IP address...

IP Address
This will normally be filled in by mIRC and is here mainly for your information. mIRC looks up your IP address and stores it in the mirc.ini file for future reference. This way it doesn't have to look it up every time you want to connect.

If mIRC is having trouble getting your IP address then you can enter this value manually and mIRC will assume that it is correct. If this value is wrong you will still be able to log on to IRC but you will not be able to initiate DCC Send/Chat sessions (you will only be able to accept them).

On connect, always get...
These options are here because of the different types of internet connections people have. Some people have a fixed Local hostname and IP address, other have a dynamic Local Hostname, others a dynamic IP address, and yet others have both. If you don't know what kind of connection you have, leave both of these checked.

Selecting **Always get Local Host** automatically turns on the **Always get IP address** option. De-selecting the **Always get IP address** option automatically turns off the **Always get Local Host** option.

Lookup Method
If you find that mIRC is not resolving your IP address correctly you might try changing from **Normal** to **Server** or vice versa, which might solve the problem.

With the **Normal** method, mIRC relies on your winsock to reply with the correct information.

With the **Server** method, mIRC looks up your local host through the IRC Server, and then performs a /dns on it to resolve it to an IP address.

The Server method will most likely be slower, you can tell when it has been completed when you see your local host name and IP address displayed in the status window.

Note: If changing the above switches **still** doesn't solve the problem, or you don't know what to enter for your local host or ip address, contact your **internet provider** or **system administrator**.

Identd

mIRC can act as an **ident server** and sends the specified **User ID** and **System** as identification. This server will be more useful to some people than others. In general it is better to leave it active as some systems might refuse a connection if there is no reply to an ident request.

Enable Ident Server
Check/uncheck this to turn the identd server on or off

User ID
This can be your account or user name on your system. For most people this will be the User ID portion of their email address (the text before the @ sign). Valid characters for a User ID are: **. 0-9 A-Z _ - a-z**

System
This identifies your operating system. For all intents and purposes, replying with a value other than UNIX would not be very useful for most people.

Port
This should usually be 113.

Show Identd requests
This displays any identd requests sent to your identd server if it is turned on.

Enable only when connecting
This turns the identd server on **only** when you're connecting to an IRC server. The moment an identd request is received and replied to, or the moment you connect to the IRC server and see the MOTD, the identd server is turned off.

Note: This server will reply to **all** ident queries sent to the specified port ie. it is not limited to replying to an IRC server for mIRC. If you have turned on the identd server and it isn't replying to identd queries then it is probable that the type of internet account you have is preventing mIRC from replying, or that another program is running which has control of the identd port.

User id from email address
Makes mIRC use the User ID from your email address in identd replies.

The /identd command
You can also use the **/identd [on|off] [userid]** command to change the above settings.

Firewall

mIRC can connect to an IRC Server through a **Socks4**, **Socks5**, or **Proxy** firewall.

The **main** purpose of this option is to allow access to IRC through a firewall at work, or more rarely through a network set up at home. The majority of home users should keep this option turned **off**.

Hostname
The address of your firewall server, can be either a named address or an IP address.

User ID
Can be the account or user name on your system. For most people this will be the User ID portion of their email address (the text before the @ sign).

Password
The password required to access the firewall.

Note: If you are connecting via a proxy, and you enter a password, mIRC only supports http proxy 'basic' authorization.

Port
This is usually 1080 for Socks firewalls.

Initiate DCCs through firewall
mIRC uses a passive protocol to establish DCC connections when a client is behind a **Socks5** firewall.

This will not work with older versions of mIRC or other IRC clients because no standard exists. You can find out more about the protocol here.

Note: Socks4 and Proxy are limited in functionality, and only allow IRC server connections.

The /firewall command
This command allows you to change the above firewall settings.

/firewall [-cmN[+|-]d] [on|off] <server> <port> <userid> <password>

The **-c** switch clears the userid and password values.
The **-mN** switch set the connection type, where N is 4 or 5 for Socks4 or Socks5, or p for proxy.
The **+|-d** switch turns dccs through a firewall on or off.

IRC

Prefix own messages
If this is selected, your nickname will prefix any messages you type in a channel/query/chat.

Show mode prefix
If turned on, mIRC will prefix nicknames in channel messages with their .@%+ mode on a channel.

Iconify query window
If someone sends you a query, the default is for the query window to open, ready for input. You can select this option to force mIRC to iconify the window preventing it from taking the focus from the window you are currently in.

Use single message window
This directs all private messages or notices from other users to one single message window. You will need to use the /msg command to reply. If you want to open a query window to a user, use the /query window.

Note: You can press the **Tab** key while in the message window to cycle through the list of nicknames who have recently messaged you.

Use query for notify nicks
If the single message window option is enabled, and a user who is in your notify list messages you, mIRC will open a query for that nickname, over-riding the single message window option.

Copy messages to query
If you're talking to a user in the single message window and then decide to open a /query to the user, this copies the conversation you've had so far from the single message window to the query window.

Whois on query
Select this to have mIRC do a /whois nickname on any person that sends you a private message. The /whois will be done the first time the query window is opened.

Auto-join channel on invite
This will make you automatically join a channel when you are invited to it. mIRC will also try to minimize the window, however this might not always work.

Rejoin channel when kicked
If you are kicked from a channel, mIRC will immediately try to rejoin the same channel. It won't close the channel window unless it finds that you can't rejoin the channel.

Rejoin channels on connect
If this switch is turned on, mIRC will automatically rejoin channel windows which are open when you reconnect to an IRC server after being disconnected.

Keep channels open
This will keep a channel window open even if you are disconnected from the server, or kicked from a channel.

Hide channel key

If this is turned on, mIRC will not display the channel key in the channel window titlebar.

Short Join/Part messages
Checking this option makes mIRC display the join and part messages in a different, more compact format.

Show user addresses
Whenever a user joins/parts/quits/is kicked/etc. from a channel, you can choose to see their address by selecting this option.

Show invites in active window
If this setting is turned off then invite messages will appear in the status window.

Show queries in active window
Shows all queries in the active channel window instead of opening up a query window. However, if you are not in a channel window, a query window will be opened.

Show notices in active window
Shows all notices in the active window instead of in the status window.

Show /whois in active window
Shows /whois results in the channel, query/chat, or custom windows if it is turned on and if a /whois is issued inside one of these windows, otherwise all /whois results are shown in the status window.

Show Ctcps in active window
Shows all Ctcp messages in the active window instead of the status window.

Show Away in active window
If you have a query window open to a user, their away message will be displayed in the query window if this option is enabled, otherwise it will be displayed in the status window.

Events...
The events dialog allows you to change the default display settings for channel-related events, such as joins, parts, etc.

Note: You can change the settings for individual channels in the channel central dialog, which can be accessed via the /channel command while in a channel.

IRC Options

Flash on...
These switches set the default flash option for any new channel or query messages that open.

Cancel away on keypress
If you set yourself as away (using /away <message>) then selecting this option cancels the away option automatically if you type a message to a channel or a query/chat window.

Skip MOTD on connect
This makes mIRC hide any MOTD information which the server sends you when you first connect to it.

Hide ping? pong! event
This hides the Ping? Pong! messages that mIRC shows in the status window whenever a server sends pings to your mIRC to check whether the connection is still active.

When closing mIRC, confirm if...
This option makes mIRC pop up a confirmation dialog asking you if you'd like to close mIRC, if any of the checked items apply.

Perform

On connect, perform these commands
You can enter a set of commands in this box which will be performed every time you connect to an IRC Server. To understand how commands work, see the Aliases help section. A quick example:

```
   /join #mIRC
```

The above command would make you join the #mIRC channel when you connect to the server.

The perform section can be turned on or off with the **/perform [on|off]** command.

Highlight

The highlight feature allows you to assign **colors** and **sounds** to incoming **messages** that contain certain **words** that you want to watch for.

By using this feature, you can do things like make messages from certain people, ie. your friends, a different color, or make mIRC play a sound if someone says a specific word.

Your highlight list displays your current list of highlight rules, which you can modify by clicking the Add, Edit, or Delete buttons.

Enable Highlighting
This turns highlighting on or off.

Highlight lines with these words
This is the list of words that you want mIRC to check for in messages from users. The words must be separated by commas.

Your words are matched against whole words, or words enclosed in non-alphabetic characters. You can also use **wildcards** *? to make a word match non-whole words, eg.: *help*

Note: You can also use Variables or Identifiers as a highlight word.

Color
This is the color that you want lines matching your highlight criteria to appear in.

Play sound
This is the sound that you want played for lines that match your highlight criteria.

Flash message
If this is turned on, mIRC will flash the mIRC icon if a match is made and your mIRC isn't the active window, or if it is minimized.

Note: You can also include variables or identifiers in the flash message.

Play Flash Sound N times
This allows you to specify how many times you want mIRC to play the flash sound.

Match on
Allows you to choose whether the match is applied to the message, the nickname, or both.

Messages

Timestamp events
This option timestamps various types of incoming/outgoing messages with the current time and date, using the format found in the $asctime() identifier.

Strip codes
This allows you to strip out the Control Codes codes from incoming private messages or channel messages.

Only if the number of codes exceeds...
This applies the strip codes feature only if the number of control codes in a message exceeds a certain number.

Ctcp finger reply
The message a user receives when they /ctcp finger you.

Quit message
The message displayed to other users when you quit IRC.

SJIS/JIS Conversion
Makes mIRC encode outgoing SJIS messages into JIS, and decode incoming JIS messages into SJIS, for Japanese systems.

Multibyte characters
This setting will improve the display, and mark/copy behaviour, of text containing multi-byte characters in mIRC windows.

Process ANSI codes
Makes mIRC interpret ANSI color codes and convert them to mIRC color codes.

Catcher

This feature looks for any text that looks like a URL or Email address in incoming messages and saves it for your future reference.

Enable address catching for...
If this option is turned on mIRC will catch references to URLs and Emails and store them in the URLs list window.

mIRC looks for URLs beginning with "http://", "ftp://", "gopher://", "www.", and "ftp.". mIRC also checks to make sure addresses are not added to a list if they already exist. Addresses longer than 256 characters are ignored.

mIRC will also catch any text that looks like an email as long as the word "mail" is mentioned in the same line of text as the email address.

On View...
When you select View from the popup menu to view a URL, mIRC can ask your browser to open a new separate URL window and it will also activate it if required.

On Send...
When sending URLs to a channel, query, etc. mIRC can send only the URL or both the URL and it's description.

Place ? marked URLs at top of list
If this is checked then mIRC will place ? marked URLs at the top of the URL list, otherwise they will be placed at the bottom of the list.

Delete All ? marked URLs on exit
To prevent your URL list getting too long you can choose to have all ? marked items deleted when you exit mIRC. Any items whose marker has been changed to something other than ? will remain in your list for future reference.

Chat links
These options enable or disable support for chat links, and allow you to specify whether you want a confirm dialog to be displayed whenever a chat link asks mIRC to connect to a server.

Hint: You can click your right mouse-button in the URL window for a popup menu which provides various URL functions.

Logging

This section allows you to changed various options relating to the logging of your conversations on IRC.

Automatically log...
This turns on automatic logging for all **channel** and **chat** windows.

Strip codes
This makes mIRC strip any Bold, Underline, Reverse, or Color control codes from text that is being logged to a file.

Lock log files
If this switch is turned off and your log files are being saved properly then you should **leave** it turned off, otherwise turn it on.

Trim log files
This option limits the size of your log files to the specified size, each time they grow beyond this maximum the oldest part of the log at the top of the file is trimmed off so that the most recent part remains.

Timestamp logs
If this is turned on, all lines in a log file are time-stamped.

Include network
If turned on, the name of irc network is included in the log filename.

If you enable the **make folder** option, a separate logs folder is created for each unique network that you use.

Date filenames
This makes mIRC create filenames which are prefix with the current date to allow you to organize your logs by date.

You can choose to have filenames dated by day, week, or month. If dated by week, log files begin on days 1, 7, 14, and 21, and if dated by month, they begin on day 1 of that month.

Channel log files dated by day are closed at 12am and re-opened with the new date.

If you turn on the **except status** switch, the status window will not have its logfiles dated.

View logs
Allows you to view your current log files.

Merge logs
Allows you to merge log files based on the filename date.

Log and Buffers folder
The folder in which all log files and buffer saves are stored.

Flood

Flood protection attempts to prevent you from **flooding** a server with messages sent in **response** to requests from other users via CTCP or a script.

Flooding usually results in your being **disconnected** from IRC servers since they place a limit on how much information you can send at one time.

mIRC will count the number of bytes you send to a server, and will initiate a flood check if you exceed a certain maximum number of bytes.

Trigger flood check after...
This is the number of bytes at which mIRC should check if it might be flooding the server or not. Setting this greater than 500 bytes isn't too helpful since that might be the maximum amount a server allows. The lower the number, the more sensitive mIRC will be, and the slower it will reply. Default 400.

Max. lines in buffer
The maximum number of lines mIRC will buffer.

Max. lines per person
The maximum number of messages a user can have have in the buffer.

Ignore person for...
How long to ignore a user who has exceeded their maximum number of buffered messages. If zero, no ignore is done.

Queue Op commands
If this is enabled, the MODE and KICK commands are also queued in the flood queue.

Queue own messages
If this is enabled, all of your own messages, such as notice and private messages, are queued in the flood queue.

Enable protection for...
This enables flood protection for ctcp replies and whois requests.

Show status updates
Displays the flood queue status a) when there is a new item in the queue, b) every 10 seconds, if the queue status changed, and c) when the queue becomes empty.

The /flood command
This command also allows you to turn flood protection on or off and to change the above settings. Typing /flood with no parameters gives you the current flood status. You can type **/flood on** to turn flood protection on with the default settings.

```
/flood [on|off|clear] <bytes> <maxlines> <maxmessages> <ignoretime>
```

```
/flood 200 10 2 30
```

Here mIRC will check for flooding if it has sent 200 or more characters to the server, will buffer a maximum of 10 lines and ignore the rest, will only allow each user 2 buffered lines, and will ignore a user for 30 seconds if that user exceeded the maximum number of buffered messages.

The flood protection method also performs intelligent queuing in order to satisfy the maximum number of users as quickly as possible, so that no single user can hog the queue.

Event Beeps

On Event, play sound
This allows you to assicuate sounds with specific events, eg. for the disconnect event, if you are disconnected by the server without having typed /quit or selected the disconnect menu item, then mIRC will beep or play the specified sound to indicate that you are no longer connected to IRC.

Beep on channel/query message
Makes mIRC beep whenever a message is sent to a channel or query window that isn't currently the active window. Note that all this option does is automatically turn on the **Beep** setting in a channel window's System Menu when it first opens. So changing this setting doesn't affect windows which are already open.

Beep on message while in buffer
Makes mIRC beep if someone speaks on a channel while you are scrolling back through lines in the scrollback buffer.

Event beep
Sets the number of event beeps that are played on an event, as well as the millisecond delay. To turn off notification all together, specify zero beeps.

Use internal beep
Makes mIRC use its own internal beep sound for events instead of the default windows sound.

Use pc speaker
Makes mIRC play beeps through your pc speaker.

Sound Requests

Sound requests allow users to share the experience of playing sounds together. When someone uses the **/sound** command on a channel, all users on the channel who have that same sound will hear it play.

Accept sound requests
If this option is turned on, mIRC will listen for **/sound** requests from other users.

On Sound Request...
If a sound is already playing and a new sound request is received, you can either have mIRC halt the currently playing sound and play the new sound, or you can choose to have mIRC ignore the new sound.

mIRC can also **warn** you if a user has requested a sound that you don't have so that you can then ask the user for that sound.

Listen for !nick file get requests
If someone sends you a message starting with an exclamation mark prefixing your nickname mIRC will assume that they are requesting a sound file from you and will search your sounds directory for the file. If it finds it, it dcc sends it to the user.

Send !nick file as private message
If this option is turned off and you send the message !nick file to a channel, everyone on the channel will see it. if you prefer not to crowd the channel with these messages, you can turn this option on and the user will receive the message privately.

Location of wave/midi files
Whenever a sound is requested, mIRC will look in these directories and all of their **subdirectories** for it. These directories are also searched when you use the /splay command.

The /sound command
You can send a sound request to another user using the /sound command.

/sound [on|off|nick/channel] <file.wav|file.mid|file.mp3> <message>

If a **nick/channel** is not specified, the message is sent to the current channel or query window.

The sound file must end in **.wav or .mid**, and may be located in the default sounds directory or in any of the subdirectories in the sounds directory. You do not need to specify a directory for the filename unless the file is not in the sounds directory.

The **message** is optional, if you do provide one it appears exactly like an action command.

Agents

mIRC supports Microsoft Agent if you have it installed on your system. An **agent** is an animated character that can speak text and perform actions.

Note: The agent dialog will only be visible if agent is already installed on your system. If it isn't installed, you can find links to related websites and resources, as well as download information, on the mIRC website at http://www.mirc.co.uk/agents.html

The agent section in the options dialog allows you to change the way agent **behaves** and to specify the **events** that you want agent to speak.

Agent Settings
This main section of the agent dialog allows you to select your default agent character, the size of the character, and various display options.

The **auto-hide** feature hides the agent whenever mIRC is **minimized**, though note that if agent needs to **speak** it will become visible again.

Agent Events
You can enable agent for **channel**, **private**, and **other** events, and more precisely specify which events you want spoken in the the Agent Events dialog.

Lexicon
The lexicon allows you to replace words or characters in spoken text with other words or characters, eg. you could replace **:)** with **smiley face**.

Speech Options
The speech settings allowing you to change the **speed**, **pitch**, and **volume** of spoken text.

The **lowercase** options change nicknames, channels, or the whole line of text into lowercase. This is useful since agent speaks **uppercase** characters as individual letters.

The **ignore message** feature stops agent speaking a line of text if the line contains a certain **percentage** of non-alphanumeric characters, ie. letters and numbers, which can happen when users play ascii text pictures.

The **only speak if channel name has changed** option stops mIRC from speaking the channel name for every message.

Mouse

The **double-click** feature allows you to specify a set of commands that will be executed whenever you double-click in a certain window.

For example, for the **channel nickname listbox** you could enter:

/query $1

which means that when you **double-click** on a nickname, a query window opens up and you can start talking with that nickname privately.

Drag Drop

The **Drag Drop** feature allows you to **pick up files** from other programs eg. a file manager, and **drop** them on either **Message** or **Channel** windows. Depending on the type of file you drop onto a window, a different action will be performed according to the commands you've defined for that file type.

You can define different actions for different types of files by **associating a command** with **a file ending**.

For example, if you wanted to mIRC to **read** a text file to another user whenever you drop files ending in **.txt** then you might make an association such as:

```
*.txt:/play $1 $2-
```

In this case, **$1** stands for the name of the user or channel where the file has been dropped, and **$2-** stands for the name of the file that was dropped. The /play command would send each line in the specified text file to the user.

You can also have different aliases executed for the same file type depending on whether you hold down the **shift key** or not when you drop the file.

Currently the **default** settings for dropping files with **no shift key** pressed are:

```
*.wav:/sound $1 $2-
*.*:/dcc send $1 $2-
```

Which means that if you drop a Wave file, it will be played with the /sound command, and if you drop any other type of file it will initiate a dcc send to that user.

The **default** settings for dropping files with the **shift key** held down are:

```
*.*:/dcc send $1 $2-
```

Which is the same as above, initiating a dcc send to the active user.

Note: If you drop a file which has a space in it, it is enclosed in "" quotes.

DCC

DCC allows you to connect **directly** to another IRC client, instead of going through the IRC Network, to **Send** and **Get** files, and to **Chat** privately over a more secure connection.

On Send Request
This option determines how mIRC behaves when someone tries to send you a file. By default, mIRC will pop up a dialog asking you if you want to accept the file, however if you select the **Auto-get file** option then mIRC will automatically accept the file. If you select **Ignore All** then all incoming dcc send requests are ignored.

If you choose to **accept** the file, mIRC will ask the sender to begin the file transfer, at which point you should begin receiving the file.

If someone tries to send you a file that already exists in your get directory then you will be shown a warning that the file exists. You then have the option to either overwrite the file, resume the transfer, or rename the file.

If you choose to overwrite the file then the whole file will be downloaded from the beginning and any existing file of the same name is erased.

If you select resume then mIRC will attempt to negotiate a transfer resume to get the remaining part of the file.

If Auto-get and File exists
In the case where you have turned on the Auto-get file option, and you already have a file of the same name in your download folder, you have the option of mIRC popping up a dialog to ask if you want to accept the file, or to automatically resume or overwrite the existing file.

Trusted Users
This section allows you to add a list of trusted nicknames, addresses, or user levels, from which mIRC will automatically accept dcc sends.

Note: Please read about the dangers of Accepting Files on IRC before you accept files.

On Chat Request
This option determines how mIRC behaves if someone sends you a chat request. By default, mIRC will pop up a dialog asking you if you would like to accept the chat request, however you can make mIRC automatically accept the chat request, or just ignore all incoming chat requests.

DCC Options

On DCC Completion
When a file transfer is completed, or a chat session ends, you can make mIRC close the window automatically, as well as beep to audibly indidate the end of a dcc session. The beep options depend on the settings in the Event Beeps dialog.

Time-out in Seconds
When a user sends you a Send or Chat request, a dialog pops up and waits for you to accept or decline. The Get/Chat Dialog time out value determines how long the dialog will wait for your reply before it closes.

During a file transfer mIRC will wait the number of seconds sepcified in the Send/Get Transfer Time Out settings for a response from the other client before closing the connection.

The Fileserver Time Out determines how long a user in a Fileserver session can remain idle before mIRC closes the connection.

DCC Ports
This option allows you to specify the range of ports that mIRC will use when making DCC connections. The usual range is 1024 to 5000, though it is possible to specify 64000 as the highest port on some systems.

Max. DCC Sends
This limits the number of simultaneous remotely initiated DCC Sends mIRC will handle.

Note: This applies to users who request a DCC Send remotely ie. via a script file or via the DCC Server, not if you initiate a DCC manually.

DCC Folders

DCC Get Folders
This allows you to specify dcc get directories where received files will be placed according to their extension. Files which don't match any of the extensions you specify are placed in the default get directory.

If you specify a **command** to be performed when a file is downloaded, the **$1-** identifier refers to the filename.

You can manually redirect an incoming dcc send to a specific folder with **/dcc get <folder>**. You can use **/dcc reject** to reject the dcc send. Both of these commands must be called from within the CTCP event or the on DCCSERVER send event. $filename returns name of file.

Ignore all files except...
This allows you to specify the types of files that you want mIRC to accept or reject automatically when someone tries to send you a file.

You can also use **/dcc ignore [on | off | only | except]** to turn this feature on or off.

Turn ignore back on
This option turns dcc ignore back on after a while if you **manually** turned it off temporarily in order to accept a file.

DCC Fileserver

Max. Fileservers
This limits the number of fileserver sessions that can be open simultaneously.

Max. DCC Gets
This limits the number of simultaneous DCC Gets a user can request.

Max. Cps
This limits the maximum cps used by all dcc sends being sent by a fileserver.

The **/dcc maxcps <N>** command can be used to changed the Max Cps value on the fly.

Note: This setting is also applied to /dcc sends initiated in a remote script. You can override this by using **/dcc send -lN** which applies no limit if N is zero.

Root Directory
This specifies the root directory that a user will see when they first enter a fileserver initiated via the DCC Server. The user will be able to access all files and directories within this root directory.

Welcome text file
This specifies the welcome text file that will be sent to a user when they first connect to a fileserver.

Display fileserver warning
If this option is checked, the fileserver warning is displayed whenver the /fserve command is used to initiate a fileserver session. To learn more about fileservers see the File Server section.

DCC Server

The mIRC **DCC Server** listens for direct connections to your IP address from other **mIRC** clients.

Enable DCC Server
This turns the DCC Server on or off.

Listen on Port
The DCC Server listens on port 59 by default, however you can change this to another port number.

Listen for...
You can have the DCC Server listen for only certain types of connections, such as DCC Sends, Chats, or Fileserver requests. For example, if you turn off the DCC Chat listen option, the DCC Server will ignore any chat requests.

Perform DNS lookup
When someone connects to your DCC Server only their IP address is available for identification. If you check the DNS lookup switch, mIRC will perform a /dns on the IP address to try to resolve it to a named address.

Note: It can take anything from a second to more than minute to resolve an address depending on network conditions, and sometimes it may not resolve at all.

You can change the settings of the DCC Server from the command line using:

/dccserver [+|-scf] [on|off] [port]

You can Send/Chat to the DCC Server using the DCC Send/Chat dialogs and specifying an **IP address** instead of a nickname.

From the command line, you can use /dcc [send|chat|fserve] with an IP Address instead of a nickname to initiate a connection to the DCC Server.

Note: /dcc fserve only works for IP address connections and does not work via IRC.

For a technical specification of the **DCC Server protocol**, see here.

Display

Show Toolbar
This options allows you to turn the toolbar and tooltips on or off.

Show Switchbar
This option turns the switchbar on or off.

Fill Switchbar
This option stretches buttons to fill the whole of the switchbar to make them easier to click.

Include DCCs
This option allows you to choose between showing DCC Send/Get windows as buttons in the switchbar or as normal icons.

Sort buttons
If this option is turned on, window buttons are sorted according to type.

Always highlight
Turning this on makes mIRC highlight buttons/icons whenever a new message appears in a window, even if the window is visible.

Multi-line bar
This doubles the height of the switchbar and displays two rows of window icons.

Blink icons
Blinks the icon in switchbar buttons for query/chat windows if there's a message/highlight event, or a flash event.

Position
You can position the switchbar at the top, bottom, left, or right of the main mIRC window.

Height and Width
These allow you to size the switchbar to your preferred height and width.

Event, Message, Highlight
Whenever a new message is displayed in a channel or query window, mIRC highlights the switchbar icon for that window with the chosen color.

The **event** color is used for all non-message events, such as joins, parts, modes, etc. The **Message** color is used for actual messages, and the **Highlight** color is used for highlighted messages, ie. messages that match your Highlight settings.

Display Options

Show this text in the mIRC titlebar
This allows you to specify text that will be shown in the mIRC title bar.

Use multi-line editboxes
This option allows you to choose between using a single line editbox, where the text scrolls to the right when you reach the end of the edit box, or a multi-line editbox, where the text starts again at the left and scrolls downwards. This only affects chat and channel windows, the status window permanently has a single line edit box.

Hint: If you type / into an editbox and press the enter key, it'll show the last line you entered in that editbox. If you type /! it will show the last line you typed in the last window you were in.

Enable dual monitor support
If you have more than one monitor connected to your computer, you can enable this option to make mIRC display dialogs, popup menus, etc. properly when it is used on different monitors.

Fast screen update
Speeds up the display of text in windows by updating less often, this might help avoid display lag in some situations.

Line spacing
This allows you to set the default line-spacing for messages in channels and chat windows.

Note: You can change the line-spacing for individual windows via the System Menu.

Line marker
Whenever you close or minimize a window, the line marker marks the spot where new messages come in to a channel or query window. This allows you to scroll back and easily see which messages you've missed.

You can press Control+L to scroll back to view the line marker. The line marker is **only** updated after you've scrolled back to view it.

Visual Styles
This dialog allows you to change the appearance of mIRC in various ways.

Transparency
If you're running mIRC under 2K/XP, you can enable **transparency** for any mIRC windows that you place on the **desktop**.

Windows

The desktop windows feature allows to you to make specific windows open as desktop windows, ie. outside of the main mIRC window. This means you can keep specific windows open on the desktop and keep mIRC minimized at the same time.

Hide minimized desktop windows
By default, minimized desktop windows appear as icons in your taskbar, turning this option on makes them minimize to mIRCs switchbar so they don't take up taskbar space.

Main mIRC window always on top
This makes the main mIRC window stay on top of all other windows.

Tray

This option only appears in the 32bit mIRC under operating systems that use a tray.

Always show mIRC icon in tray
Turning this option on makes mIRC always display an mIRC icon in the tray, even when it isn't minimized.

On startup minimize mIRC to tray
This option makes mIRC automatically minimize to the tray when it is first run.

Place mIRC in tray when minimized
If this option is turned on, the mIRC window is hidden when you minimize mIRC, and a small mIRC icon appears in the tray.

If this option is turned off, you can force mIRC to minimize to the tray by holding down the Shift key when you minimize mIRC.

If you hold down the Shift key when you quit mIRC, the next time you run it, it will be minimized.

If you right-click on the mIRC icon in the tray, a popup menu listing your current channels/queries appears; those that have a checkmark next to them have had their icons highlighted due to incoming messages. You can click a menu item to open the window.

If you double-click on the mIRC tray Icon and mIRC is the active window, it will be minimized, if it's already minimized, it will be restored.

Animate icon when there is activity
This option animates the mIRC tray icon by flashing the 'i' and 'r' letters when there is activity, and by turning the icon purple when it is connected, as well as showing a revolving planet when trying to connect to an IRC Server.

Location and name of icon to use in tray
This allows you to specify a different icon in place of the standard mIRC icon in the tray.

/tray -iN <filename>
This command changes the mIRC Tray icon to the Nth icon in the specified file (which might be an .exe, .dll, or .ico).

Note: The animate icon option can't work if you've selected a custom icon.

General

Command Prefix
The default standard prefix to a command is a **/** character, however you can specify another character if the **/** key is hard to access on your keyboard.

Note: Regardless of the character you choose here, mIRC still recognizes the / character and uses it internally.

Window buffer
This limits the scrollback buffer to the specified number of lines. Note that if a scrollback buffer already contains more than the specified number of lines it will be shortened. You can always use the **/clear** command in a window to clear the scrollback buffer completely. If you are scrolling back through the buffer, lines will not be removed until you return to the bottom of the buffer.

Line separator
You can specify a line separator to be used in the status window. You can use a space to have a blank line. If you leave the box empty, lines in the status window will not be separated.

ESCape key minimizes window
To make a window minimize whenever you press the ESCape key, enable this option.

Control+K pops up color index
If this is turned on, mIRC will pop up a color index dialog whenever you press Control+K in an editbox to insert a color code.

Hotlinks only when shift key is pressed
By default, when you move the mouse over an address/nickname/channel/etc. the mouse turns into a Hotlink pointer. By turning this option on, the hotlink will only appear if you have the shift key pressed.

Right-click in listbox selects line
If turned on, this feature makes mIRC select the line under the mouse when you right-click in a listbox.

Titlebar right-click needs shift key
If turned on, you have to hold down the shift key when you click your right mouse button on a window titlebar to open/close the window.

Control+Tab uses switchbar order
This changes the normal Control+Tab behaviour when switching windows so that it follows the current window order on the switchbar.

Tab key changes editbox focus
This makes it easier to switch to the **second editbox** in a channel window. The tab key still works as nick/channel/etc. completion if the cursor is next to a word. If enabled, you need to use **Shift-Tab** instead of **Tab** to cycle between the list of nicks that messaged you.

DDE

mIRC supports **DDE** communication, which allows external applications to control mIRC or to request information from it. Note that it's also possible to communicate with mIRC directly by using SendMessage().

mIRC's default service name is **mIRC**, though note that this can be changed via the options dialog.

DDE Topics

mIRC supports the following DDE Topics.

For each of the following DDE topics an example is given using mIRC's /dde command and $dde() identifier (both are explained after this section) which should help you understand how they work.

Topic: **COMMAND**
Transaction Type: XTYP_POKE
Item: None
Data (Arguments): One line of text containing the command to be performed.
Returns:Nothing
Example: /dde mirc command "" /server irc.dal.net
Description: This tells mIRC to execute whatever command you give it.

Topic: **EVALUATE**
Transaction Type: XTYP_REQUEST
Item: The line of text containing variables or identifiers that you want evaluated
Data (Arguments): None
Returns:Evaluated line

Topic: **CONNECT**
Transaction Type: XTYP_POKE
Item: None
Data (Arguments): one line of text in the form: address,port,channel,number
Returns:Nothing
Example: /dde mirc connect "" irc.undernet.org,6667,#mIRC,1
Description: This will tell mIRC to connect to the server irc.undernet.org, port 6667 and after it has connected it will automatically join channel #mIRC. If the number at the end is a 1 then the mIRC window will be activated, if it is a 0 it will not.

Topic: **CONNECTED**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns:"connected" if you're connected to a server, "connecting" if you're in the process of connecting to a server, and "not connected" if you're not currently connected to a server.
Example: //echo mIRC is currently $dde(mirc, connected) to a server

Topic: **EXENAME**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: The full path and name of the mIRC EXE file. eg. "c:\mirc\mirc.exe"
Example: //echo The mIRC exe path and name is $dde(mirc, exename)

Description: This might be useful for applications that need to know the path and name of the exe file.

Topic: **VERSION**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: mIRC version info.
Example: //echo mIRC's version number is $dde(mirc, version)

Topic: **INIFILE**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: The full path and name of the main INI file that is being used. eg. "c:\mirc\mirc.ini"
Example: //echo mIRC's main ini file is $dde(mirc, inifile)
Description: This might be useful for applications that need to take a look at the INI file for various bits of information.

Topic: **NICKNAME**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: The nickname currently being used.
Example: //echo My mIRCbot is using the nickname $dde(mirc, nickname)

Topic: **SERVER**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: The server to which you were last or are currently connected. eg. "irc.undernet.org"
Example: //echo My mIRCbot's irc server is $dde(mirc, server)

Topic: **PORT**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: The port currently being used to connect to the irc server
Example: //echo My mIRCbot is using port $dde(mirc, port)

Topic: **CHANNELS**
Transaction Type: XTYP_REQUEST
Item: None
Data (Arguments): None
Returns: A single line of text listing the channels which you are currently on separated by spaces. eg. "#mirc #mircremote #irchelp"
Example: This should only be called by an application, not from within an mIRC alias.
Description: The line of text that this returns could end up being quite long, maybe several k's worth of text, so an application should be prepared to handle this.

Topic: **USERS**
Transaction Type: XTYP_REQUEST
Item: Channel name, in the form #channel
Data (Arguments): None

Returns: A single line of text listing the users on the specified channel separated by spaces.
Example: This should definitely only be called by an application, not from within an mIRC alias.
Description: You can only request the names of users on a channel which mIRC has joined. The line of text it returns could be very, very long, so any application that uses this must be prepared to handle a single line of several k's worth of text.

DDE Commands and Identifiers

The following commands and identifiers allow you to communicate with other DDE-enabled applications.

All DDE transactions are **synchronous**, mIRC waits at most one second for a reply or acknowledgement to any DDE request that it makes.

/ddeserver [[on [service name]] | off]
mIRC's default service name is mIRC. The mIRC DDE Server defaults to on at startup unless it finds another mIRC or another application using it's current DDE Service name.

You can use the /ddeserver command to manually turn the dde server on or off. If you have more than one mIRC acting as a DDE server then you should specify a different **service name** for each mIRC.

/dde [-re] <service> <topic> <item> [data]
This sends an **XTYP_POKE** by defaults unless you specify the **-r** switch in which case an **XTYP_REQUEST** is sent, or if you specify the **-e** switch, an **XTYP_EXECUTE** is sent.

If you are sending an **XTYP_POKE** then all four arguments are mandatory.

If you are sending an **XTYP_REQUEST** then the first three arguments are mandatory. This is why you can see a **""** in the examples above, it acts as a filler and isn't actually used for anything.

If you are sending an **XTYP_EXECUTE** then only the first three arguments are required.

$dde(name, topic, item, delay)
Returns the value returned by the specified service name, topic, and item, by sending an XTYP_REQUEST.

//echo My other mIRC is $dde(mirc, connected) to $dde(mirc, server)

The **item** and **delay** parameters are optional.

The **delay** value indicates that you want $dde() to wait N seconds for a reply before giving up. The default is one second, which is usually enough.

Note: If the value returned by $dde is too long for mIRC to handle, $dde returns a value of $error.

$isdde(name)
Returns $true if the specified dde name is in use, $false otherwise.

$ddename
Returns the current dde service name in use by the mIRC DDE Server.

SendMessage

Applications can now use **SendMessage()** to communicate with the 32bit mIRC.

Performing Commands

The following call to SendMessage() makes mIRC **perform the commands** that you specify.

SendMessage(mHwnd, WM_MCOMMAND, cMethod, 0L)

mHwnd - the handle of the main mIRC window, or the handle of a Channel, Query, etc. window.

WM_MCOMMAND - which should be defined as WM_USER + 200

cMethod - the way in which you want mIRC to process the message, where:
1 = As if typed in editbox
2 = As if typed in editbox, send as plain text
4 = Use flood protection if turned on, can be or'd with 1 or 2

Returns - 1 if success, 0 if fail

Evaluating Identifiers and Variables

The following call to SendMessage() makes mIRC **evaluate the contents** of any line that you specify.

SendMessage(mHwnd, WM_MEVALUATE, 0, 0L)

mHwnd - the handle of the main mIRC window, or the handle of a Channel, Query, etc. window.

WM_MEVALUATE - should be defined as WM_USER + 201

Returns - 1 if success, 0 if fail

Mapped Files

The application that sends these messages **must** create a **mapped file** named **mIRC** with **CreateFileMapping()**.

When mIRC receives the above messages, it will open this file and use the data that this mapped file contains to perform the command or evaluation. In the case of an **evaluation**, mIRC will output the **results** to the mapped file.

The mapped file must be at least **1024** bytes in length.

Finger Server

The finger server listens for **finger requests** on port 79.

Enable finger server
This turns on the finger server.

Show finger requests
If this is turned on, finger requests are displayed in the status window.

Finger file
You can specify a finger file, which must be a plain text file, and this must be set up with **named sections** which will be used to reply to any finger requests. Each section begins in the following way:

```
[name]
line1
  .
  .
  .
lineN
```

The section name corresponds to the **userid** that is being fingered. eg. if someone fingers khaled@mirc.com, then I will have a section named [khaled].

There should be at least one section named **[default]** which will be used to reply to a finger request which does not specify a user, or specifies a user that doesn't exist.

Therefore, for my own system I would have two sections:

```
[default]
etc.

[khaled]
etc.
```

This can, for example, allow you to setup a menu system, with various sections that can answer user finger queries, the default section being the main menu.

If you want identifiers or variables in a line in the finger text file to be evaluated, you must prefix the text line with a $ character.

On a related note, check out the /finger command.

Lock

The Lock dialog allows you to lock various features in mIRC.

Ask for Password
If this is turned on and you have set a password by using the **Lock** button, mIRC will ask you for a password each time it is run.

The Lock Button
This allows you to set a password which locks the options in the lock dialog.

Note: If you hold down the **Control** key when you **minimize** mIRC, it will ask you for the password when you try to open the mIRC window later. If you use Control+Minimize with no password set, mIRC will ask you for a **temporary** password.

Disable...
This allows you to disable features relating to Sending/Recieving files and the Fileserver.

You can also disable **private messages** and **dcc chats**, useful if you want to limit conversations to public channels.

Disable commands
This allows you to disable the **/run** and **/dll**. Although these commands can be quite useful in scripts, they are disabled by default to protect new users since **some** scripts might be harmful. See the Accepting Files section for more information. Once you are confident enough and know how to accept files safely, you can enable these if you wish.

Limit channels...
This feature allows you to limit the channels that mIRC can join when on IRC. mIRC will only be able to join the channels specified in this list. mIRC will also only list these channels when you try to retrieve the full channels list via the List Channels dialog.

Hide tray menu...
This hides the list of channels/queries in the tray menu when mIRC is locked and minimized to the tray.

Basic IRC Commands

IRC commands allow you to perform specific functions on IRC, such as maintaining control of a channel and the users on it. The following list of **Basic** IRC commands should help you get started. There are also Other Commands you can look at later.

General Commands

/JOIN #channel
Join the specified channel.

example: /join #irchelp

This will make you join the #irchelp channel. Once on a channel, anything you type will be seen by all the users on this channel. The #irchelp channel is very useful, so say hello and then ask any questions you want. If the channel you specified doesn't exist, a channel with that name will be created for you.

Some channels may also have keys ie. a password, which you need to specify when using the /join command.
example: /join #irchelp trout

This will make you join the #irchelp channel using "trout" as the password.

/PART #channel
Leave a channel.

example: /part #irchelp

/LIST [#channel] [-MIN #] [-MAX #]
Lists currently available channels. You can also tell mIRC to show only channels with a minimum and a maximum number of people. If you specify a #channel then mIRC will only list information for that channel. If you specify wildcards, eg. *love* then mIRC will list all channels that contain the word **love** in them.

example:	/list
example:	/list -min 5 -max 20
example:	/list #mirc
example:	/list *love*

/ME message
Tells the current channel or query about what you are doing.

/MSG nickname message
Send a private message to this user without opening a query window.

/QUERY nickname message
Open a query window to this user and send them a private message.

/WHOIS nickname
Shows information about someone.

/NICK nickname
Changes your nickname to a new nickname.

/QUIT [reason]
This will disconnect you from IRC and will give the optional message as the reason for your departure. (this message only appears to people who are on the same channels as you).

example: /quit That's all folks!

/AWAY [away message]
Leave a message explaining that you are not currently paying attention to IRC. Whenever someone sends you a MSG or does a WHOIS on you, they automatically see whatever message you set. Using AWAY with no parameters marks you as no longer being away.

example: /away off to get something to eat, back in a moment!

/TOPIC #channel newtopic
Changes the topic for the channel.

example: /topic #friendly Oh what a beautiful day!

/INVITE nickname #channel
Invites another user to a channel.

Channel and User Commands

If you have Op status, the following commands give you control over both a channel and the users on it.

/KICK #channel nickname
Kicks named user off a given channel.

example: /kick #gb Ed

/MODE #channel|nickname [[+|-]modechars [parameters]]
This is a powerful command that gives channel operators control of a channel and the users on it.

Channel modes

ModeChar	Effects on channels
~~	~~~~~~~~~~~~~~~~~~~~
b <person>	ban somebody, <person> in "nick!user@host" form
i	channel is invite-only
l <number>	channel is limited, <number> users allowed max
m	channel is moderated, (only chanops can talk)
n	external /MSGs to channel are not allowed
o <nickname>	makes <nickname> a channel operator
p	channel is private
s	channel is secret
t	topic limited, only chanops may change it
k <key>	set secret key for a channel

User modes

ModeChar	Effects on nicknames

~~	~~~~~~~~~~~~~~~~~~~~~~
i	makes you invisible to anybody that does not know the exact spelling of your nickname
o	IRC-operator status, can only be set by IRC-ops with OPER
s	receive server notices
v	gives a user a voice on a moderated channel

Here a few examples of the MODE command:

To give someone Op status: /mode #channelname +o nickname

Giving someone Op status means giving them control over the channel and the users on it. Give this out sparingly and to people you trust.

To op several people: /mode #channelname +ooo nick1 nick2 nick3

To de-op someone: /mode #channelname -o nickname

To ban someone: /mode #channelname +b nickname (or user address)

example: /mode #animals +b Jiminy
example: /mode #tree +b joe@bloggs.edu

To Unban someone: /mode #channelname -b nickname (or user address)

example: /mode #gb -b Ed

To Make a channel invite only: /mode #channelname +i

You must now **invite** a user for them to be able to join your channel.

There many more commands but this list should help you get started. To learn more about IRC commands you should download an IRC FAQ.

mIRC Commands

The following commands are mostly unique to mIRC, though some are only modifications or extensions of standard IRC commands.

Note: To view the full list of commands see the Search dialog.

/ajinvite [on | off]
Turns auto-join on invite on or off.

/alias [filename] <aliasname> <command>
Adds, removes, replaces aliases; it is limited to single line aliases and will not affect mutiple line definitions.

/alias /moo /me moos!

This will replace the first matching alias with the new command. To remove an existing aliases:

/alias /moo

To add an alias to a specific alias file, you would use:

/alias moo.txt /moo /me moos!

If you don't specify a filename, it defaults to using the first filename in which the alias exists, or if it doesn't exist then it uses the first loaded aliases file.

/amsg <message>
This and the **/ame** command send the specifed message or action to all channels which you are currently on.

/anick <nickname>
Changes your alternate nickname.

/background [-aemsgdluhcfnrtpx] [window] [filename]
Changes the background picture setting for a window. This can also be changed via a windows System Menu.

-a = active window
-m = main mIRC window
-s = status window
-g = finger window
-d = single message window

-e = set as default

-cfnrtp = center, fill, normal, stretch, tile, photo

-l = toolbar
-u = toolbar buttons
-h = switchbar

You can right-click in the toolbar/switchbar to pop up a menu for changing these settings.

Toolbar buttons can use RGB Color **255,0,255** for transparency, the BMP must be of the same form as that in mIRC resources. It should be a 16 or 256 color BMP.

-x = no background picture

Note: The **window** name should only be specified if none of the window switches are specified. The **filename** does not need to be specified if you are only changing the display method.

/ban [-ruN] [#channel] <nickname|address> [type]
Bans someone from the current channel using their address. To do this, it first does a /userhost on the user, which gives it the user's address, and then it does a /mode # +b <user address>.

If you specify the **-uN** option then mIRC pauses N seconds before removing the ban.

If you specify the **-r** switch then /ban **removes** the ban of the specified type for that nickname, eg. /ban -r nick 2

If you do not specify a ban type, then mIRC uses the whole nick!*user@host to do the ban. If you are banning an IP address then a wild card replaces the last number of the IP address. If you are on the channel then the #channel specification is not necessary.

If you specify a wildcard address it is used as-is, if you specify a full address then the type mask is applied to it.

For a list of ban types see the $mask identifier.

Note: This command uses the IAL maintained by mIRC.

/beep <number> <delay>
Beeps a number of times with a delay.

/channel [#channel]
Pops up the channel central window for the channel window you're currently in. You can also specify a #channel name to open the channel central for a channel you've already joined but which isn't the active window.

/clear [-sghlc] [windowname]
Clears the buffer of the current window. If you specify a window name, that window's buffer will be cleared.

The **-s** switch clears the status window.
The **-g** switch clears the finger window.
The **-l** switch clears the side-listbox in a custom window.
The **-c** switch clears the click history in a picture window.
The **-h** switch clears the editbox command history for a window.

You can use **/clearall** to clear the buffers of all windows.

/clipboard [-a] <text>
Copies the specified text to the clipboard. The **-a** switch makes it append the text to any existing text in the clipboard.

/close [-icfgms@] [nick1] ... [nickN]

Closes **all** windows of the specified type and the specified nicknames. If **no** nicknames are given, **all** windows of the specified type are closed. The type of window is denoted by c for chat, f for fserve, g for get, i for inactive dcc windows, m for message (query), s for send, and @ for custom windows.

You can also specify the Nth window for -cfgs by appending a number, eg. /close -s4 nick, would close the 4th open dcc send to nick.

/color [-r] <name> <index>
Allows you to change the color settings for items in the Colors dialog.

You can change the color of a box in the color dialog, you can use /color <index> <rgb>.

You can also reset the Nth color box to its default RGB value with /color -r <N>.

/copy -ao <filename> <filename>
Copies a file to another filename or directory. You can also use wildcards for the source filename, and a directory name for the destination. The **-o** switch overwrites a file if it exists. The **-a** switch appends the first file to the second one.

/creq [+m|-m] [ask | auto | ignore]
This is the command line equivalent of setting the DCC Chat request radio buttons in the dcc options dialog (see /sreq below). The +m|-m switch turns the minimize setting on|off.

/ctcpreply <nick> <ctcp> [message]
Sends a reply to a ctcp query.

/ctcpreply goat HELP no help available.

/debug [-cnpt] [N] [on | off | @window | filename]
Outputs raw server messages, both incoming and outgoing, to a debug.log file, or a custom @window.

/debug -n @moo, opens a custom @window minimized
/debug -c off, turns off debugging and closes the associated custom @window
/debug -pt, wraps or timestamps messages
/debug N @moo, uses color N for messages

The **$debug** identifier returns the name of debug file or @window.

Note: /debug works independently for each server connection.

/describe <nick|channel> <message>
Sends an action to the specified nickname or channel, the same as the /me command, except that /me is used while in a query or channel window so you don't need to specify the target when using it.

/disconnect
Forces a disconnect from a server. This is different from the /quit command which sends a quit message to the server and waits for the server to disconnect you.

/dll <name.dll> <procname> [data]
This allows you to call routines in a DLL designed to work with mIRC.

/dns [-ch] [nick|address]
Resolves an address. If mIRC sees a "." in the name you specify it assumes it's an address

and tries to resolve it. Otherwise it assumes it's a nickname and does a /userhost to find the users address and then resolves it. If you give it an IP address, it looks up the host name.

You can queue multiple /dns requests, and you can view the current queue by using /dns with no parameters.

The **-c** switch clears all currently queued DNS requests, except for the one currently in progress.

The **-h** switch forces /dns to treat the parameter as a hostname.

Note: Due to way the DNS lookup works, any DNS related functions currently in progress eg. connecting to a server, must be resolved before subsequent requests. This means that if a prior DNS is having problems resolving, subsequent DNSs have to wait until it times out before they can be resolved.

/dqwindow [on|off|show|hide|min]
Manipulates the single message window.

/ebeeps [on | off]
Enables or disables the sounds in the event beeps dialog.

/echo [color] [-deghiNtsaqlbfnmr] [#channel|[=]nick] <text>
Prints text in the specified window using the specified color (0 to 15).

/echo 3 #mIRC Testing

would print "Testing" in the color green in channel window #mIRC, assuming it's already open.

If a channel/nickname isn't specified, the **-s** switch echos to the status window, the **-d** switch echos to the single message window, and the **-a** switch echos to the currently active window.

The **-e** switch encloses the line in line separators.
The **-iN** switch indents the wrapped line by N characters.
The **-h** switch forces lines to hard-wrap so resizing the window doesn't change the line.
The **-t** switch prefixes the line with a timestamp if global time stamping is on or timestamping is on for that window.
The **-q** switch makes it not display the text if called from an alias using the . prefix.
The **-l** switch makes it apply the highlight settings to the line that's displayed.
The **-bf** switches make it apply the beep/flash settings in the window it is echoing to.
The **-n** switch prevents the echo from hiliting the window switchbar icon.
The **-m** switch indicates that the line should treated as a user message, not an event.
The **-g** switch prevents the line from being logged to the log file.
The **-r** switch applies the strip settings in the messages dialog.

Note: This text is only displayed in your own window, it isn't sent to the server, no one else can see it.

/editbox [-sanop|[=]window] <text>
Fills the editbox of the current window with the specified text.

The **-s** switch specifies the Status window.

The **-a** switch specified the Active window.
The **-p** switch indicates that a space should be appended to text.
The **-n** switch fills the editbox and presses the enter key in the editbox.
The **-o** switch applies the command to the second editbox in a channel window.

To specify a dcc chat window, prefix the nickname with an **=** equal sign.

/emailaddr <address>
Changes the email address in the Connect dialog.

/exit
Close down mIRC and exit.

/filter [-asgdfwxnpriocteubglL] [n-n2] [c s] <infile | dialog id> <outfile | dialog id> [alias] <matchtext>
This command scans lines of text in a window or file and if any of them contain matchtext, they are written out to another window or file which you can then use.

The **infile** can be a filename or a window name (custom or normal). The **outfile** can be a filename or a custom window name. You should specify the **-fw** switches if the names are ambiguous eg.

/filter -ff in.txt out.txt *mirc*

This indicates that both are filenames, and:

/filter -wf #in.txt #out.txt *help*

indicates that the first is actually a window name, and the second is a filename.

The **-a** switch **sorts** filtered lines by calling the optional **[alias]**. The alias is passed two lines in $1 and $2, it must compare these and return -1, 0, or 1 to indicate relative sort order of these lines to each other.

The **-x** switch excludes matching lines.
The **-n** switch prefixes lines with a line number.
The **-s** switch makes the status window the infile.
The **-g** switch makes the finger window the infile.
The **-d** switch makes the single message window the infile.
The **-p** switch wraps the text output in a custom window.
The **-r** switch specifies the range of lines **n to n2** for filtering.
The **-b** switch strips BURK codes when matching text.
The **-g** switch indicates that matchtext is a regular expression.

The **-i** switch indicates that you have provided a [dialog id] custom dialog control as the input.
The **-o** switch indicates that you have provided a [dialog id] custom dialog control as the output.

The **-c** switch clears the output window/file before writing to it.

The **-t** switch sorts the output based on [c s], column C using character S as the columns separator
The **-e** specifies a descending sort, and **-u** a numeric sort.

The **-l** switch filters from the side-listbox in the first window, and **-L** filters to the side-listbox in the second window.

You can filter blank lines by specifying **$crlf** for the matchtext.

This command also fills the **$filtered** identifier with the number of matches found, if any.

Note: If the input and output are the same window/file, mIRC will process the request correctly.

/findtext -n <text>
This searches active window for the specified text (same as Control+F).

/flash [-wbrN] [window] <text>
This flashes the specified mIRC window/icon with **text** in the titlebar but **only** if mIRC is not the active application.

The **-b** switch makes mIRC beep every second.
The **-w** switch makes mIRC play the Flash sound specified in the Event Beeps section.
The **-rN** switch makes mIRC repeat the flash only N times.

/flushini <filename>
Flushes the specified INI file to the hard disk. INI files are cached in memory, so you may want to do this to make sure that your INI is updated properly.

/font [-asgbd|window] <fontsize> <fontname>
This allows you to change the font for the current window. If no parameters are specified, the font dialog pops up, otherwise the specified parameters are used. You can make the font bold by using the **-b** switch.

The **-a** switch applies the setting to the active window, **-s** to the status window, and **-g** to the finger window.

The **-d** switch makes the font the default for that type of window, eg. for all channels, or all chats.

Note: If you use a negative number for the font size, it will match the size of fonts in the font dialog.

/fullname <name>
Changes the full name in the connect dialog.

/help [keyword]
Brings up the section in the mIRC help file which matches the specified keyword.

/hop [-cn] [#channel] [message]
Parts the current channel and joins a new one. If no new channel is specified, it parts and rejoins the current channel without closing the window.

The **-c** switch cycles the specified channel by parting and rejoining it.

The **-n** switch minimizes the channel window.

/join [-inx] <#channel>
This is a standard IRC command for joining a channel.

The **-i** switch makes you join the channel to which you were last invited.
The **-n** and **-x** switches minimize/maximize the channel window when you join it.

/linesep [-s|window]
Prints the line separator selected in the Options dialog in the specified window.

/links [-nx]
Retrieves the servers to which your current server is linked.

The **-n** and **-x** switches minimize/maximize the window when it opens.

/load <-a|-pscqnm|-ruvsN> <filename>
Loads the specified alias, popup, or script.

/load -a aliases.ini	loads an aliases file
/load -pc status.ini	loads a channel popup
/load -pn status.ini	loads a nickname list popup
/load -ru users.ini	loads a users file
/load -rv vars.ini	loads a variables file
/load -rs script.ini	loads a scripts file

If you try to load a file that is already loaded, it's contents are updated and it's position in the alias/script processing order is maintained.

You can also use the **/reload** command with the same parameters to reload a file without triggering the on start/load events in the script being loaded.

If you specify the **N** with /load -rsN, this loads/reloads the script into the Nth position in the script list.

Note: You can only load one section at a time.

/loadbuf [lines] [-pirsgleopcNt<topic>] <window | dialog id> <filename>
Loads the specified number of lines from the end of the file of filename into the specified window.

/loadbuf 20 @test info.txt

This loads the last 20 lines of info.txt into custom window @test.

/loadbuf 10-40 @test info.txt

This loads lines 10 to 40 of info.txt into custom window @test.

The **-p** switch forces lines of text to wrap when added to the window.
The **-i** switch makes sure that lines are indented if they wrap.
The **-r** switch clears the contents of the output window.
The **-s** and **-g** switches apply the command to the status and finger windows respectively.
The **-l** switch applies the command to the side-listbox in a custom window.
The **-e** switch evaluates variables and identifiers in the line being read.
The **-cN** switch specifies the default color for lines.

The **-t** switch loads the text under the [topic] section in an INI or plain text file.

The **-o** switch indicates that you have specified [dialog id] parameters instead of a window name in order to load text into a custom dialog control.

/localinfo -uh [host ip]
Looks up and sets your local info settings. The **-u** switch performs a /userhost lookup, the **-h** switch does a normal lookup. If you wish, you can also set the local info manually by specifying the host and ip values.

/log <on|off> <window> [-f filename]
Turns logging on and off for a window, if you specify a filename the logs file dialog is not popped up.

/mdi -act
Allows you to arrange icons, and cascade/tile windows.

/mkdir <dirname>
Creates the specified directory.

/nick <nickname>
Changes your nickname.

/omsg [#channel] <message>
This and the **/onotice** command sends the specified message to all channel ops on a channel. You must be a channel operator to use these commands. If the #channel isn't specified, then the current channel is used.

Commands P to Z

mIRC Commands

The following commands are mostly unique to mIRC, though some are only modifications or extensions of standard IRC commands.

Note: To view a full list of commands see the Search dialog.

/partall [message]
Parts all of the channels you are currently on. On certain IRC Servers, you can also specify a message.

/pdcc [on | off]
If turned on, tries to speed up dcc sends by sending packets ahead of acks.

/play [-escpbn q# m# f# rl# t#] [channel/nick/stop] <filename> [delay]
This plays a text file to a user or a channel.

/pop <delay> [#channel] <nickname>
Performs a delayed op on a nickname. The purpose of this command is to prevent a channel window filling up with op mode changes whenever several users have the same nickname in their auto-op section.

mIRC will pause around <delay> seconds before performing the op. If <delay> is zero, it does an immediate op. Before performing the op it checks if the user is already opped. If you do not specify the #channel, the current channel is assumed.

/pvoice <delay> [#channel] <nickname>
Works the same way as the /pop command except that voices a user.

/query [-n] <nick> [message]
Opens a query window to the specified nickname. If a message is provided, it is sent.

If the **-n** switch is specified, the window is opened in a minimized state.

/queryrn <nick> <newnick>
Changes the nickname of an open query window.

/raw [-q] <command>
Sends any parameters you supply directly to the server. You **must** know the correct RAW format of the command you are sending. Useful for sending commands which mIRC hasn't implemented yet. The -q switch makes the raw work quietly without printing what it's sending. This command does the same thing as **/quote** in other IRC clients.

/raw PRIVMSG nickname :Hellooo there!

/remini <inifile> <section> [item]
Deletes whole sections or single items in an INI file.

/remini my.ini DDE ServerStatus

This would delete the ServerStatus item, and:

/remini my.ini DDE

Would delete the DDE section.

See the /writeini command below for a related example.

Warning: Do not use this command to modify any of the INI files currently being used by mIRC.

/remove [-b] <filename>
Deletes the specified file.

The **-b** switch deletes the file and moves it to the recycle bin.

/rename <filename> <newfilename>
Renames a file, can also be used to move a file from one directory to another.

/resetidle [seconds]
This resets the $idle identifer to zero or to the number of seconds you specify.

/rmdir <dirname>
Deletes the specified directory.

Note: If the directory contains files, it cannot be deleted.

/run [-np] <filename> [parameters]
Runs the specified program with parameters.

The **-n** switch minimizes the window of the application being run.
The **-p** switch sets the working path to the path of the application being run.

You can enclose the filename or parameters in quotes if you need to. If you specify a **non-executable** file, mIRC tries to open it with the application associated with that file.

/save <-pscqnm|-ruv> <filename>
Saves the specified popup or remote users/variables file.

/save -ps status.ini	saves the status popup to status.ini
/save -pn nick.ini	saves the nickname list popup to nick.ini
/save -ru users.ini	saves the user list to users.ini
/save -rv vars.ini	saves the variables list to vars.ini

Note: You can only save one section at a time.

/savebuf [-sgao] [lines] <window | dialog id> <filename>
Saves the specified number of lines from the end of the buffer of the specified window into the specified filename.

/savebuf 20 @test info.txt

This saves the last 20 lines in custom window @test to info.txt.

/savebuf 10-40 @test info.txt

This saves lines 10 to 40 in custom window @test to info.txt.

The **-s** switch saves the status window buffer, the **-g** switch saves the finger window buffer, and the **-a** switch makes it append the text to the end of a file instead of overwriting it.

The **-o** switch indicates that you have specified [dialog id] parameters instead of a window name in order to save text from a custom dialog control.

/saveini
Updates all mIRC-related INI files with the current settings.

/say <message>
This lets you define an alias that writes directly to a channel as if you were saying something. So "/say Hello there" would be the same as just typing "Hello there". This is useful in an alias when you want to ask the same question (or send the same information) again and again.

/info /say Please note that the games server is currently down and will be offline for a few hours...

Note: You can't use this command in the remote section. Use /msg #channel <message> instead.

/server [-mnsar] <server/groupname> [port] [password] [-i nick anick email name] [-j #channel pass]
Connects you to a server, first disconnecting you from the current server.

/server irc.undernet.org 6667 mypassword

If you type /server with no parameters, mIRC will connect to the last server you used. If you use the server command while still connected, you will be disconnected with your normal quit message and will then connect to the specified server.

You can also use /server N which connects to the Nth server in the server list in the connect dialog.

You can also use /server groupname which will cycle through all the servers in the server list which have that group name until it connects to one of them.

The **-m** switch creates a new server window for that connection and connects to the server. The **-n** switch does the same thing but does not connect to the server.

If you specify any of the -sar switch, the format of the command becomes:

/server -sar [server] [-p port] [-g group] [-w password] [-d description]

-s sorts the servers list
-a adds a server. If it exists, it is updated
-r removes a server

mIRC tries to find a match for either the server **address** or the **description** in the existing servers list. You can also specify **none** for -g -w and -d to clear the current setting.

/showmirc -nrstxop
Manipulates the display of the main mIRC window, where -n = minimize, -r = restore, -s = show, -t = tray, -x = maximize, -o = on top, -p = not on top.

/sline [-a|r] <#channel> <N|nick>
Selects or deselects lines in a channel nickname listbox. It can select either the Nth

nickname in a listbox, or a specified nickname.

If you do not specify any switches, any existing selections in the listbox are cleared. If you specify the **-a** switch then the specified is selected without affecting the selection states of other lines. If you specify the **-r** switch then the specified item is deselected.

/speak <text>
Sends the specified text to Monologue (or Text Assist) which is a program that speaks whatever text is sent to it.

Note: This feature only works with the very old versions of the above software. The new versions do not support the method that mIRC uses.

/splay [-cwmpq] <filename>
Plays the specified sound, see the Playing Sounds section.

/sreq [+m|-m] [ask | auto | ignore]
This is the command line equivalent of setting the DCC Send request radio buttons in the dcc options dialog (see /creq above). The +m|-m switch turns the minimize setting on|off.

/strip [+-burc]
Turns control code stripping options in Options dialog on/off.

/strip +bur-c

would turn bold, underline, reverse stripping **on**, and turn color stripping **off**.

/timer[N/name] [-ceomhipr] [time] <repetitions> <interval> <command>
Activates the specified timer to perform the specified command at a specified interval, and optionally at a specified time.

If you are not connected to a server and you start a timer, it defaults to being an offline timer which means it will continue to run whether you are connected to a server or not.

If you are connected to a server and you start a timer, it defaults to being an online timer, which means that if you disconnect from the server, it will be turned off. You can specify the **-o** switch to force it to be an offline timer.

/timer1 0 20 /ame is AWAY!

Timer1 will repeat an all channel action every 20 seconds until you stop the timer.

If you specify a delay of 0 seconds, the timer will trigger immediately after the calling script ends.

/timer5 10 60 /msg #games For more info on the latest games do /msg GaMeBoT info

Timer5 will repeat this message to channel #games every sixty seconds and stop after 10 times.

/timer9 14:30 1 1 /say It's now 2:30pm

This will wait until 2:30pm and will then announce the time once and stop.

To see a list of active timers type /timers. To see the setting for timer1 type /timer1. To deactivate timer1 type /timer1 off. To deactivate all timers type /timers off. If you are activating a new timer you do not need to specify the timer number, just use:

/timer 10 20 /ame I'm not here!

And mIRC will allocate the first free timer it finds to this command.

If you specify the **-c** switch, this makes mIRC "catch up" a timer by executing it more than once during one interval if the real-time interval isn't matching your requested interval.

If you specify the **-m** or **-h** switch, this indicates that the interval delay is in milliseconds.

Note: The **-h** switch creates a high-resolution multimedia timer. This type of timer should **only** be used in critical timer situations since it uses system resources heavily.

If you specify the **-e** switch, this executes the command associated with the specified timer name, also works if you specify a wildcard name.

The **$ltimer** identifier returns the number of the timer that was just started by the /timer command.

Instead of using a number you can also specify a **name** for a timer.

/timershow 0 10 echo -a $nick $server $time

You can force identifiers to be re-evaluated when used in a /timer command by using the format $!me or $!time.

If you wish to turn off a range of timers, you can use a wildcard for the number, for example:

/timer3? off

Will turn off all timers from 30 to 39.

The **-pr** switches pause and resume a timer respectively.

The **-i** switch makes a timer dynamically associate with whatever happens to be the active connection. If a server window is closed, the timer is associated with the next available server window.

/timestamp [-fs|a|e] [on|off|default] [windowname]
Turns time-stamping of events **on** or **off**. If you specify **default**, uses the global timestamp setting.

-s = for status window
-a = for active window
-e = for every window

If a windowname is not specified, then the global timstamp switch is turned on or off.

The **-f** switch allows you to set the timestamp format, eg. /timestamp -f [HH:nn]

/titlebar [@window] <text>
Sets the main application titlebar. If you specify a custom @window name, then the titlebar

for that custom window is changed.

/tnick <nickname>
Changes your nickname to a temporary nickname, without affecting your main or alternate nicknames.

/tokenize <c> <text>
Fills the $1 $2 ... $N identifiers with tokens in <text> separated by character <c>, eg.:

/tokenize 44 a,b c,d,e

The above command would set $1 = a, $2 = b c, $3 = d, $4 = e

/unload <-a|-nrs> <filename>
Unloads the specified alias or remote script file.

/unload -a aliases.ini	unloads the alias.ini file
/unload -rs script.ini	unloads the script.ini file

The **-n** switch prevents a script from having the on unload event triggered.

Note: You can only unload one script at a time.

/updatenl
Usually the channel nicknames list and IAL in a kick/part/quit script event are updated after the script finishes, this command updates them immediately.

/url [on | off | show | hide | -dran] [[N | mark] | address]
Show or hides the Url list window, and allows you to modify the current list of addresses in it.

The **-r** switch deletes the Nth item, or all items that match the **mark** you specify.

The **-an** switches allow you to open a browser window to an address, where -a = activate browser, and -n = use a new browser window.

/winhelp <filename> [key]
Opens a help file with the specified search key.

/write [-cida l# s# w#] <filename> [text]
Writes lines to a text file. For example:

/write store.txt This line will be appended to the end of file **store.txt**

The **-c** switch clears the file completely before writing to it, so it allows you to start with a clean slate.

/write -c c:\info.txt This file will be erased and have this line written to it

The **-l#** switch specifies the line number where the text is to be written.

/write -l5 c:\info.txt This line will overwrite the 5th line in the file

The **-i** switch indicates that the text should be inserted at the specified line instead of overwriting it. If you do not specify any text then a blank line is inserted. If you do not specify a line number then a blank line is added to the end of the file.

/write -il5 c:\info.txt This line will be inserted at the 5th line in the file

The **-d** switch deletes a line in the file. If you don't specify a line number then the last line in the file is deleted.

/write -dl5 c:\info.txt

The above command will delete the 5th line in the file.

The **-s#** switch scans a file for the line beginning with the specified text and performs the operation on that line.

/write -dstest c:\info.txt

This will scan file info.txt for a line beginning with the word "test" and if found, deletes it.

If you do not specify any switches then the text is just added to the end of the file.

The **-w#** switch scans a file for the line containing the specified wildcard text and performs the operation on that line.

Note: With both -s# and -w# you can enclose the scan text in quotes if it contains spaces.

The **-a** switch indicates that mIRC should append the line of text you specified to the existing text of the specified line.

Note: You cannot use this command to write to an INI file. If you do so, you will most likely corrupt the INI file.

/writeini -n <inifile> <section> <item> <value>
Writes to files in the standard INI file format.

If the **-n** switch is specified, mIRC will attempt to write to the .ini file even if it is larger than 64k.

A part of the mirc.ini file looks like this:

```
[DDE]
ServerStatus=on
ServiceName=mirc
```

You could achieve this with /writeini by using:

```
/writeini my.ini DDE ServerStatus on
/writeini my.ini DDE ServiceName mirc
```

You can delete whole sections or items by using the /remini command.

Warning: Do not use this command to modify any of the INI files currently being used by mIRC.

Commands A to O

Aliases

mIRC allows you to create aliases and scripts to speed up your IRC session or to perform repetitive functions more easily. To create aliases you must know some IRC commands.

Aliases can be called from the **command line**, from **other aliases**, and from **popup** and **remote** scripts. An alias **cannot** call itself recursively mainly because this seems to cause more problems for users than it solves.

Examples

The following examples show you how to create aliases that perform simple functions.

/gb /join #gb

If you now type **/gb** this is the same as typing **/join #gb**.

/j /join $1

We have now added a **parameter** string. If we type **/j #gb** this is the same as typing **/join #gb**. The **$1** refers to the first parameter in the line that you supply.

/yell /me $2 $1

If you now type **/yell There! Hello** the action command will be **/me Hello There!** The number after **$** specifies the number of the parameter in the string that you entered.

/jj /join $?

The **question mark** indicates that you should be asked to fill in this parameter. The parameter you supply will be inserted in the line at that point. So if you type **/jj** a dialog will pop up asking you for the channel you want to join. If you enter **#gb** then the final command will be **/join #gb**.

/jj /join #$1

the **# sign** indicates that the parameter you specify should be **prefixed** with a hash indicating that it is a **channel**.

/jj /join $?="Enter channel to join:"

This does the same thing but now the dialog will have the "Enter channel to join:" line displayed inside it.

/aw /away $?="Enter away message:" | /say $!

This is similar to the line above except for the addition of the **$!** parameter. This refers to the text you **just** typed into the parameter box. ie. the away message. This saves you having to type the same message twice.

/give /me gives $$1 a $$2

The **double $$** means that this command will **only** be executed if a parameter is specified. If you specify only one parameter in the above command it will **not** be executed. You can also do **$$?1** or **$?1** which means try to fill this value with parameter one if it exists. If parameter one doesnt exist, ask for it. In the first case the parameter is necessary for the command to be executed, in the second case it isn't.

/slap /me slaps $1 around with $2-

The **$2-** indicates that everything following and including **parameter 2** should be appended to the command line. if you type **/slap Sheepy a large trout** the final line will be **/me slaps Sheepy around with a large trout**.

You can also specify **$2-5** which means use only parameters 2 to 5.

/laugh /me laughs at $1's joke

Anything appended to a **$** parameter is appended to the final parameter. So if in the above example we type **/laugh mimi** the final command would be **/me laughs at mimi's joke**.

/silly /say Hel $+ lo th $+ ere $+ !

Parameters are normally separated by a space. To make mIRC **combine** parameters you can use the **$+** identifier. The above line will say **Hello there!**.

/p /part #

The **# sign** refers to the **channel** you are currently on. So if you are on channel #blah, and you type **/p** then mIRC replaces the # sign with #blah, and the final command is **/part #blah**.

/op /mode # +o $1

To op someone you can now just type **/op goat** instead of the whole /mode command.

/dop /mode # -ooo $1 $2 $3

You can now deop three users by typing **/dop goat mike bongo**.

For **multiple commands** you should use a **|** character (the shifted character usually under the \ key). So to write an alias that kicks and bans someone:

/dkb /kick # $1 | /mode # +b $1

The [ ] evaluation brackets

If you want greater control over the order of evaluation of identifiers, you can use the **[ ] brackets**. Identifiers within these brackets will be evaluated first, from left to right. You can **nest** brackets.

/say % [$+ [$1]]

You can also **force** a previously evaluated identifier to be re-evaluated by using extra [ ] brackets.

/set %x %y
/set %y Hiya!
/echo [[%x]]

The { } brackets

You can create **multi-line** scripts by using the **{ } brackets**. This allows you to create an alias which performs several commands.

```
/poem {
   /msg $1 The Wendigo, the Wendigo,
   /msg $1 It's eyes are ice and indigo...
}
```

The If-then-else statement

You can use if-then-else statements to decide which parts of your script executes based on the evaluation of a comparison.

```
/number {
   if ($1 == 1) echo One
   elseif ($1 == 2) echo Two
   else echo Unknown number!
}
```

This creates an alias which tests if the parameter you supplied is the number 1 or the number 2.

For more information, see the if-then-else section.

The Goto command

The /goto command allows you to **jump** from one point in a script to another point.

```
/number {
   if ($1 == 1) goto one
   elseif ($1 == 2) goto two
   else goto unknown
   :one
   echo One
   halt
   :two
   echo Two
   halt
   :unknown
   echo Unknown number!
   halt
}
```

Using a **goto** incorrectly could lead to an **infinite loop**. You can **break** out of a currently running script by pressing **Control+Break**.

Note: I didn't prefix the above commands with the **/** command prefix. This is because the command prefix is really only needed when entering a command on the command line. In scripts, all lines are assumed to start with a command, so you don't need to use the / command prefix.

While Loops

Repeats a loop containing a set of commands while the expression in brackets is true.

```
var %i = 1
while (%i <= 10) {
   echo 2 %i
   inc %i
}
```

The expression in brackets uses the same format as an if-then-else statement.

Multiple while loops can be embedded. You can use **/break** to break out of the current loop, and **/continue** to jump to the beginning of the loop.

The Return command

The /return command halts a currently executing script and allows the calling routine to continue processing.

You can also optionally specify a return value which will be stored in the $result identifier. The $result can then be used in the calling routine.

```
/return [value]
```

The Halt command

The /halt command halts a script and prevents any further processing. You can use this in remote scripts to prevent mIRC from replying to normal ctcp messages, or in aliases to halt an alias, and any calling aliases, completely.

Identifiers and Variables

An **Identifier** returns the value of a built-in mIRC variable. For example, $time would return the current time. Whenever mIRC finds an identifier in your command, it replaces it with the **current** value of that identifier.

For a list of identifiers, see the Identifiers section.

Variables are identifiers whose values you can create and change yourself and use later in your scripts.

For more information on variables, see the Variables section.

Custom Identifiers

A custom identifier is just an **alias** which returns a value, and you can use that aliases name with an identifier prefix.

For example, create an /add alias such as:

```
add {
   %x = $1 + $2
   return %x
}
```

And then use it in a command:

```
//echo Total is: $add(1,2)
```

You can supply as many parameters as you want, ie. $add(1,2,...,N).

You can also use the **$prop** identifier to refer to your own custom properties:

```
add {
  %x = $1 + $2
  if ($prop == negative) return $calc(-1 * %x)
  return %x
}
```

//echo Total is: $add(1,2).negative

Note: Built-in identifiers of the same name have priority.

Remote Scripts

You can add aliases to remote scripts by using the **alias** prefix and then entering your alias as usual.

```
alias add {
  %x = $1 + $2
  return %x
}
```

This is the same custom identifier as above, except it uses the **alias** prefix.

If you specify the **-l** switch in the alias definition, the alias becomes accessible only by commands in the same script and invisible to the command line and other scripts.

```
alias -l add {
  %x = $1 + $2
  return %x
}
```

Function Key support

You can **redefine** function keys to perform certain commands, just like aliases. For example:

```
/F1 /say Hello!
/sF2 /query $1
/cF3 /ctcp $1 version
```

The **s** and **c** prefixes for **Shift** key and **Control** key respectively.

Note: A function key will behave **differently** depending on the window in which it is used. For example, when using it in a **query window** the $1 parameter refers to the selected users nickname. If you're on a **channel** and the **nickname listbox** is active then the function key will work on the selected nicknames. If the listbox is **not** active, the function key will just work on the channel.

Command prefixes

If you are executing a command from the command line ie. by typing it into an editbox, you can **force** mIRC to **evaluate** identifiers in that command by prefixing it with two **//** instead of one /. For example:

/echo My nickname is $me

Would print out "My nickname is $me" and would not evaluate the $me.

//echo My nickname is $me

Would print out "My nickname is Pengy" if your nickname was Pengy.

If you want to force a command to perform **quietly** ie. without printing out any information, then you can prefix it with a "." fullstop. For example:

```
/ignore somenick
```

Would print out information telling you that you are now igoring "somenick". If you don't want this information to be displayed, then you can use:

```
/.ignore somenick
```

If you want to perform a command without it being processed as an **alias**, you can prefix it with a ! exclamation mark.

Comments

You can add **comments** to your scripts by using the **; semi-colon** at the start of a line.

```
;This is a comment
```

You can place comments anywhere in a script, they are ignored during processing.

The $& identifier

This identifier allows you to break up a single line into multiple lines which are combined when the script is performed, so you can edit long commands more easily:

```
longline {
   echo This is an example of a long $&
   line that has been split into multiple lines $&
   to make it easier to edit
}
```

Popups

mIRC allows you to create custom popup menus for the status window, query/chat windows, channel windows, the channel nickname listbox, and main menubar. To create these you must know how to use IRC commands, how to create Aliases, and how to use Identifiers and Variables.

If you click the right mouse-button in a window, the popup menu for that window will appear and you can select menu-items which you have defined to perform certain tasks, such as opping a user or joining a channel.

Examples

Popup menu definitions use the format:

```
<menuitem>:<commands>
```

Get Help:/join #irchelp

The words before the ":" colon are the name of the menuitem. The words after the ":" colon are the commands that are to be performed. In this case, the menuitem you would see is "Get Help". The command that would be performed if you select this menuitem would be "/join #irchelp", as if you had typed it.

The format of the commands follows precisely the same as those in normal aliases. See the Aliases section to learn about aliases.

To create a **Submenu**, use a "." fullstop.

```
Join a Channel
.Get IRC help!:/join #irchelp
.Visit the folks at #friendly:/join #friendly
.Who shall we join?:/join $?
```

In this case, the name of the submenu is "Join a Channel". All the commands following it beginning with a "." are part of this submenu.

To create **menus within a submenu**, you just add more fullstops:

```
Channels
.Help
..irchelp:/join #irchelp
..mIRC:/join #mirc
..newbies:/join #newbies
.Other Channels
..Visit #friendly:/join #friendly
..Wibble Wobble:/join #wibble
.Join?:/join #$$?="Enter a channel name:"
```

To **separate** menuitems, you can use a single "-" dash on a line by itself.

```
whois ?:/whois $?
-
Misc
.Edit Temp:/run notepad.exe temp.txt
```

```
.say?: /say $?
.action?:/me $?
Names
.#irchelp: /names #irchelp
.#friendly: /names #friendly
.names ?:/names $?
-
channel list:/list
-
Join a Channel
.Get IRC help!:/join #irchelp
.Visit the folks at #friendly:/join #friendly
.Who shall we join?:/join $?
```

To use the popup menu for a channel nickname listbox, you need to select a nickname before the menu will pop up. Here is a simple nickname listbox popup menu:

```
Who Is?:/whois $1
-
Modes
.Op:/mode # +o $1
.Deop:/mode # -o $1
.Kick, Ban:/kick # $1 | /ban $1
-
DCC Send:/dcc send $1
DCC Chat:/dcc chat $1
-
Slap!:/me slaps $1 around a bit with a large trout
Query:/query $1 Hey you! hello? are you there...?
```

If you want to create a popup menu item which performs **several commands**, you can use the { } brackets. See the Aliases section for more information on how to use these.

```
Cookie {
  if ($1 == $me) echo I give myself a cookie!
  else echo I give $1 a cookie!
}
```

The above menitem can be used in the channel nicknames listbox. The $1 refers to the nickname of the user you have selected in the listbox. In this case, it checks to see if I've selected my own nickname, and if so displays the first message, otherwise it displays the second message.

The popup menus for the **Query/Chat** and the **MenuBar** work the same way as the channel listbox popup menu.

Identifiers and Variables

Variables or identifiers that are a part of a the title of a menu definition are evaluated each time the popup appears. This allows you to create popup menus that vary in appearance. If an entire menu item evaluates to $null, it is not displayed.

Remote Scripts

You can place **menu** definitions in your remote scripts by using the **menu** prefix.

```
menu status {
  Server
  .Links:/links
  .Lusers:/lusers
  .Motd:/motd
  .Time:/time
}
```

This definition would add a submenu to your status window popup menu.

You can also specify channel, query, nicklist, and menubar as the menu name, and this will add menuitems to your current popup menus for each of these windows.

```
menu nicklist {
  Slap
  .Haddock:/me prods $1 with a haddock
}
```

This definition would add a submenu to your channel nickname listbox popup menu.

You can also specify popup menus for custom windows by specifying the custom window name.

```
menu @test {
  dclick:/echo double-click!
}
```

The **dclick** item allows you to specify a command that will be performed when you double-click in a custom window listbox. You can also refer to $1 which holds the line number of the line that was double-clicked.

You can also specify **multiple** window names for a menu, eg.:

```
menu @dogs,@cats,@goats {
   dclick: /echo double-click in $active
   close: window -c $active
}
```

You can use the **$menu** identifier to refer to the menu that is about to pop up or that is assocaited with the script being performed. This allows you to modify the form of the popup based on whether it's a query, channel, etc. popup menu:

```
menu query,nicklist {
   $iif($menu == nicklist,Op):/mode # +o $$1
}
```

Menu Styles

You can place a check mark or create a disabled menu item by using the $style(N) identifier, where N = 1 for checked, N = 2 for disabled, and N = 3 for both. The $style(N) identifier must be the first word in the menu definition.

```
menu status {
  $iif($server == $null,$style(2)) Server Info
  .Motd:/motd
  .Time:/time
}
```

The above definition creates a submenu in your status window popup which is only enabled when you are connected to an IRC server.

$submenu($id($1))

This identifier allows you to dynamically create a list of menu items, and can only be called from a popup menu definition.

It calls **$id($1)**, where $id() is the name of your identifer, and where $1 = 1, and increases by 1 with each call, **adding** whatever is returned by $id() to the popup menu.

The **value** that $id() returns must be a one line definition format for a popup menu.

The iteration **ends** when $id() returns no value.

```
menu status {
   Animal
   .$submenu($animal($1))
}

alias animal {
   if ($1 == begin) return -
   if ($1 == 1) return Cow:echo Cow
   if ($1 == 2) return Llama:echo Llama
   if ($1 == 3) return Emu:echo Emu
   if ($1 == end) return -
}
```

The **begin** and **end** values are sent to check if the item should be enclosed in separators.

Note: You can't use this to create nested submenus, it will only build one single submenu.

Remote

The **remote** allows you to create scripts that react to IRC Server events, such as when a user joins a channel or sends you a message. This tool is the most complex part of mIRC and to use it you must already know how to use IRC Commands, how to create Aliases, and how to use Variables and Identifiers.

The remote consists of **three** distinct sections:

The **Users** section, where user addresses with assigned access levels are listed. Each User in your Users section can be assigned one or more levels. These access levels dictate which events a user will be able to access.

The **Variables** section, where the currently active variables are listed.

The **Scripts** section, where the scripts that you create are listed. You can load **multiple** scripts which work **independently** of each other. This means that a single IRC Server event can trigger events in **one or more** scripts. Scripts consist of **events** which can only be triggered by **users** who have the **required** access levels. You can also place Aliases in your scripts by using the **alias** prefix, and menus in your scripts by using the **menu** prefix.

Since Access Levels play an important part in the way scripts work, you should read about them carefully before proceeding. You should also take a look at remote Commands, Identifiers, and the Internal Address List, and the section on how to Halt default text being displayed if you want to display your own custom messages for events.

All of the following **Events** use the same general format except for the **ctcp** and **raw** events. The sections below provide you with **information** on each event as well as **examples** and **tips** on how to use them.

Action	Error	Mp3End	Se
Active	Exit	Nick	Se
Agent	FileRcvd	NoSound	Sig
AppActive	FileSent	Notice	Sn
Ban	GetFail	Notify	Sta
Chat	Help	Op	Tex
Close	Hotlink	Open	Top
Connect	Input	Part	Un
Ctcp	Invite	Ping	Un
CtcpReply	Join	Pong	Un
DccServer	KeyDown	PlayEnd	Us
DeHelp	KeyUp	Quit	Voi
DeOp	Kick	Raw	Wa
DeVoice	Load	RawMode	Wa
Dialog	MidiEnd	SendFail	
Dns	Mode	Serv	

Here is an example script that shows you how you can place aliases, popups, and events in a single file making it easier to distribute your scripts to other people.

Note: You should never load or use a script that you don't understand.

Remote Commands

The following commands are used to modify settings in scripts and in the remote section.

General Commands

/ctcps [on|off]
This switches processing of ctcp events on/off.

/events [on|off]
This switches processing of named events on/off.

/dlevel <level>
This changes the default user level to the specified level.

/raw [on|off]
This switches processing of numeric events on/off.

/remote [on|off]
This switches processing of all scripts on/off.

Group Commands

These commands allow to turn groups on and off in remote scripts. You can find out about groups in the Access Levels section.

/enable <group1 group2 ... groupN>
This enables the specified groups in all scripts.

/enable #one #two #three

You can also specify a wildcard to enable all matching groups:

/enable #help*

/disable <group1 group2 ... groupN>
This disables the specified groups in all scripts.

/disable #one #two #three

You can also specify a wildcard to disable all matching groups:

/disable #help*

/groups [-e|d]
This displays a list of either all, enabled, or disabled groups in your scripts.

User and Level Commands

You can use the following three commands to add and remove users to the users list, as well as to add and remove levels from existing users.

/auser [-a] <levels> <nick|address> [info]
This adds the specified nick/address exactly as it is given to the users list with the specified levels. If you specify [-a], then if the user already exists, the specified levels are added to the current levels the user has. Remember, if the first level is not preceeded by an equal

sign then it is a general access level.

/auser 1,2,3 Nick

This adds this nick with these access levels to the user list (replacing an existing user of the same name).

/auser -a 1,2,3 Nick

This adds the specified levels to this user. If the user doesnt exist, it is created.

/auser -a =1,2,3 Nick

This looks like the above command, however the =1 is very important. The =1 means that the initial general access level is **not** replaced. If you had used 1 then the initial access level would be replaced.

The **info** parameter allows you to append text to the entry that is added to the users list, you can reference this later with the $ulist() identifier.

/flush [-l] [levels]
This clears the remote user list of nickname definitions that are no longer valid.

/flush 1,2,3

For each nick in the remote user list that matches the specified levels mIRC checks to see if that nick is on any of the channels that you are currently on. If not, the nick definition is removed from the remote user list. If you do not specify levels then mIRC clears all nicks from the remote user list that don't exist on channels you are on.

You can use the **-l** switch to remove only the specified levels from entries in the user list, instead of removing the entries.

/guser [-a] <levels> <nick> [type] [info]
This works the same as the /auser command except that it looks up address of the specified nick and adds it to the user list. It does this by doing a /userhost on the given nickname, and returning an address in the format specified by type. If no type is specified then a default address format is selected.

The **info** parameter allows you to append text to the entry that is added to the users list, you can reference this later with the $ulist() identifier.

/iuser <nick | address> [info]
This allows you to set or remove the **info** appended to a user list entry.

/ruser [levels] <nick | address> [type]
If used without specifying levels, this removes the specified user from the user list. If you specify levels then these levels are removed from the current access levels of this user. If all a user's levels have been removed, the user is removed. If you specify a type then the users address is looked up with a /userhost and any users in the users list matching this address are removed.

/ruser Nick
/ruser 1,2,3 Nick
/ruser 1,2,3 Nick 1

If you use /ruser Nick! (with an exclamation mark at the end), it removes all users with an address beginning with Nick!.

/rlevel [-r] <levels>
This removes all users from the remote users list with the specified general access level.

```
/rlevel 20
/rlevel =10
```

If the -r option is specified, then this applies to all the access levels for a user (not just the first general access level). Any matching levels are removed. If a user has no more levels left then the user is also removed.

```
/rlevel -r 1,5,7,8
```

/ulist [<|>] <level>
This lists users which have the specified access levels.

/ulist <10	lists users with access levels less than or equal to 10
/ulist >5	lists users with access levels larger than or equal to 5
/ulist 4	lists users with access level 4

Note: The /guser and /ruser commands do a /userhost on a nick to find the nick's address, thus they are delayed commands since they need to wait for a reply from the server. mIRC tries to get around this delay by maintaining it's own Internal Address List which will speed things up in certain situations.

Remote Identifiers

You can use the following identifiers in scripts to refer to values relating specifically to events. There are also quite a few other identifiers which relate only to specific events, these are described with the events themselves in other parts of the help file. There are also other identifiers which can be used in both remote and non-remote scripts.

$1-
You can use the $1 $2 ... $N identifiers to refer to individual parameters in a line. You can also use $N- to refer to parameters N and onwards, and $N-M to refer to parameters $N through to $M. So to refer to a whole line, you would use $1-.

$0
Returns the number of space-delimited tokens in the $1- line.

$(...)
This identifier can only be used in the **text match** section of an **event** definition. It allows you to create a match parameter dynamically.

```
on 1:TEXT:$(*hello $me $+ *):?:echo hello back!
```

$address
Returns the address of the user associated with an event in the form user@host.

$chan
Returns the name of the channel for a specific event. For all non-channel events $chan will be $null.

$clevel
Returns the matching event level for a triggered event.

$dlevel
Returns the current default user level.

$event
Returns the name of the event that was triggered.

$fulladdress
Returns the full address of the user triggering an event in the form nick!user@host.

$group(N/#)
Returns the the name or status of a #group in a script.

Properties: status, name, fname

$group(0)	returns the number of groups
$group(1)	returns the name of the first group
$group(1).status	returns on or off for the first group
$group(#test)	returns on or off for group #test
$group(#test).fname	returns the script filename in which the group exists
$group(3).name	returns the name of the 3rd group

$maddress
Returns the address that was matched for the triggered event.

$matchkey
Returns wildcard matchtext that was used in the matching remote event.

$mode(N)
Returns the Nth nick affected by a channel mode change.

Properties: op, deop, ban, unban, voice, devoice, help, dehelp

The properties can be used to specify which type of mode change you want to check.

$mode(0).op returns the number of opped nicks
$mode(1).op returns the first opped nick

Note: This identifier is only used in conjunction with the on OP/DEOP etc. events.

$nick
Returns the nickname of the user associated with an event.

$numeric
Returns the numeric for the matching numeric event.

$script
Returns the filename of the currently executing remote script.

$script(N/filename)
Returns the filename for the Nth loaded script file. If you specify a filename, it returns $null if the file isn't loaded.

$script(0)	return the number of script files loaded
$script(2)	returns the filename of the 2nd loaded script file
$script(moo.txt)	returns $null if the file isn't loaded, or moo.txt if it is.

Note: This can't be used to reference the users or variables files.

$scriptdir
Returns the directory of the currently executing remote script.

$scriptline
Returns line number in current script.

$site
Returns the portion of $address after the @ for the user associated with an event in the form user@host.

$target
Returns the target of an event.

$ulevel
Returns the user level that was matched for the currently triggered event.

$ulist(nick!userid@address,L,N)
Returns the Nth address in the Users list that matches the specified address and level.

Properties: info

You can specify a wildcard address or a * to match any address in the user list. If you don't specify a full address, it completes the address with wildcards. If you don't specify N, the first matching address is returned.

If you specify L, only matching addresses that contain the specified level are returned.

Note: L and N are optional, but if you specify L, you must specify N.

$wildsite
Returns the address of the user who triggered an event in the form *!*@host.

Access Levels

Access levels are assigned both to a user and to an event and serve to limit a user's access to only certain events.

The **default access level** is 1 for users that are not listed in the Users list. All users can access level 1 events. The higher a user's access level is, the more events that user can access. You can change the default user level to allow unlisted users to access more commands.

Users

In the Users section you can specify a **list of users** and their access levels using the format:

<level1,level2,...,levelN>:<useraddress>

3,5,6:goat!khaled@mirc.com

The **first** level is a general access level, which means that the user can access all levels equal to or less than 3. All the **other** levels are levels that an event must specifically have to allow a user to access it.

If you want to **force** the first access level to be a specific level instead of a general access level, you can prefix it with an equal sign.

=3,5,6:goat!khaled@mirc.com

Now this user has access specifically to level 3, 5, and 6 event and to no other events.

Events

In general the format of an event is:

<prefix> <level>:<event>:<window>:<commands>

ctcp 1:HELP:*:/msg $nick No help is available for level 1 users

The above ctcp command can be accessed by all users because it is a level 1 command. So if a user with nickname goat sends you a /ctcp yournick HELP, your script will send them the above reply.

Only the **highest level** matching event is triggered for a user.

Named Levels

You can also used **named levels** which work the same way as a specific level but are easier to understand and read than a number.

friend:goat!khaled@mirc.com

on @friend:JOIN:#mIRC:/mode $chan +o $nick

This treats the word **friend** as a specific access level and matches the user with the event, and because the user is your friend, you give him ops.

Limiting Access

You can limit access to an event by specifying a special prefix which determines how an event is processed or triggered by users.

The + prefix
You can limit an event to users with a specific access level by using the + prefix.

```
10:goat!khaled@mirc.com
```

```
ctcp +5:HELP:*:/msg $nick You have accessed a level +5 event
```

The above user can't access this ctcp event even though he has an access level higher than 5 because the event is limited only to level 5 users.

The * prefix
You can allow any user to trigger an event regardless of their access level by using the * prefix.

```
on *:TEXT:help:#:/msg $nick you have accesed a * level event
```

The ! prefix
You can prevent an event from being triggered if it was initiated by you by using the ! prefix.

```
ctcp !2:HELP:*:/msg $nick You have accessed a level 2 event
```

You would be unable to access the above event regardless of your access level.

The @ prefix
You can limit events to being executed only when you have Ops on a channel by using the @ prefix.

```
10:goat!khaled@mirc.com
```

```
on @2:JOIN:#mIRC:/mode $chan +o $nick
```

When the above user joins channel #mIRC and you have Ops on #mIRC, the associated /mode command will be executed, in this case giving the user Ops. If you don't have Ops, the event will not trigger.

The & prefix
You can prevent an event from being triggered if a previous script used **/halt** or **/haltdef** to halt the display of default text for an event by using the & prefix.

```
on &1:TEXT:*:?:/echo this event won't trigger if $halted is true
```

The = suffix
You can prevent users with higher access levels from accessing **all** lower access level events by using the = suffix.

```
10:goat!khaled@mirc.com
```

```
ctcp 2:HELP:*:/msg $nick You have accessed a level 2 event
ctcp 5:HELP:*:=
```

The above user can't access any of these events because the level 5 event prevents him from accessing all HELP events with access levels lower than 5.

The ! suffix

You can prevent commands for a certain event level from being processed by using the ! suffix.

```
ctcp 5:PING:*:echo PING!
ctcp 5:*:*:!
```

The ! at the end of the line tells the remote to halt any further processing of level 5 commands.

Creating Groups

You can create separate groups in scripts by using the # hash prefix.

```
#group1 on
...
[ list of events ]
...
#group1 end
```

You can use the /enable and /disable commands to enable or disable groups. A group that is disabled will be ignored when processing events. A disabled group looks like this:

```
#group1 off
...
[ list of events ]
...
#group1 end
```

You cannot have groups within groups.

Order of definitions

Many of the prefixes and controls are sensitive to numerical order of the definitions. The safest thing is to order your definitions starting with the lowest access levels first and increasing numerically down the list, this makes it easier to keep track of which events should trigger first.

Halting default text

mIRC displays its own default text for various types of IRC Server events, such as users joining or parting a channel, however you can modify or suppress this by using a script.

The ^ event prefix

You can prevent the default text for an event from being shown by using the ^ prefix in an event definition. This allows you to show your own custom event messages.

```
on ^1:JOIN:#:echo $chan Joins: $nick | halt
```

This line is triggered by a JOIN event and shows your own custom join event message, /halt prevents the normal message from being shown.

The ^ events **don't replace** your existing events; your normal events are independent and are **still** processed whether there is a ^ event in a script or not.

If you only want to halt the default text without /halting the entire script, you can use the **/haltdef** command.

You can check if a script has already halted the default text by using the **$halted** identifier; it returns $true if a user has used /halt or /haltdef in a ^ event, and $false if not.

The ^ event prefix currently works **only** on the following types of events: ACTION, BAN, CHAT, DEHELP, DEOP, DEVOICE, HELP, INVITE, JOIN, KICK, MODE, NICK, NOTICE, OP, OPEN, PART, PING, TEXT, UNBAN, USERMODE, VOICE, QUIT, SERV, SERVERMODE, SNOTICE, TOPIC, WALLOPS.

Note: Halting the default text for an event affects how mIRC displays the most basic information about IRC events to a user, so it should be used carefully.

Example Script

The following example script shows you how you can place related aliases, popups, and events in a single file making it easier to distribute a whole script to other people.

```
;Moo Script v1.0 - contains moo related functions

;This menu definition adds a submenu to your channel popup menu

menu channel {
   Moo
   .happily:/describe # moos happily
   .woefully:/describe # moos woefully
   .philosophically:/describe # MUs
   .colorfully:/describe # moos in several hues
}

;These add aliases for shortcuts to often used messages

alias how /msg $1 How now brown cow?
alias moo /sound moo.wav moooos

;This adds a ctcp command which reacts to a moo ctcp from someone

ctcp 1:moo:*:/notice $nick Sorry, I'm all out of moos right now.

;These add events which react to specific words said on a channel

on 1:text:*moo*:#:/msg $chan okay, who let the cow loose?
on 1:text:*grass*:#:/describe $chan dribbles hungrily

;These add join and part events which react to a user joining/parting
;the channel #moo

on 1:join:#moo:{
   /msg $nick Welcome $nick to channel #moo!
   /msg $nick This is a herd-oriented channel, there are calfs present!
   /msg $nick Please refrain from profaine mooing and/or bleating
   /msg $nick Mammals enaging in such acts will be promptly demooted
}

on 1:part:#moo:/msg $nick Thanks for grazing with us on #moo!

;The following line is processed while you're doing a channels list. It
;prints to the status window any channel name/topic that has the
;word moo in it

raw 322:*moo*:/echo -s $2-
```

on ACTIVE/APPACTIVE

The **on ACTIVE** and **on APPACTIVE** events trigger when a window in mIRC is activated or when mIRC's active status changes respectively.

Format: on <level>:ACTIVE:<*#?=!@>:<commands>
Example: on 1:ACTIVE:#mirc:/echo Channel window #mIRC has been activated

Examples

on 1:ACTIVE:*:echo Activated: $active De-Activated: $lactive

The above event triggers whenever a window in mIRC is activated. The **$active** identifier returns the name of the currently active window, and the **$lactive** identifier returns the name of the window that was just de-activated.

Note: It's possible for either or both $active and $lactive to return $null.

on 1:APPACTIVE:echo mIRC active status: $appactive

The above event triggers whenever mIRC's active status has changed. The **$appactive** identifier returns $true if mIRC is active, or $false if it isn't.

on AGENT

The **on AGENT** event triggers when an Agent has finished speaking.

Format: on <level>:AGENT:<commands>
Example: on 1:AGENT:/echo Agent $agentname has finished speaking

Examples

on 1:AGENT:/echo Agent $agentname has finished speaking

The above event triggers when an Agent has finished speaking. The **$agentname** identifier returns the name of the agent associated with the event.

on BAN/UNBAN

The **on BAN** and **on UNBAN** events trigger when a user on a channel is banned or unbanned.

Format: on <level>:BAN:<#[,#]>:<commands>
Example: on 1:BAN:#mirc,#irchelp:/msg $nick Sorry but you're not allowed on $chan

Examples

on 9:BAN:#newbies:/mode $chan -o $nick | /mode $chan -b $banmask

This triggers when someone bans a user with access level 9. **$banmask** refers to the banmask used to ban the user.

on 1:UNBAN:#:/msg $bnick You have just been unbanned

This triggers when any user is unbanned from any channel. **$bnick** refers to the banned users nickname, however this would actually only be filled if the banmask itself includes a nickname. If the banmask does not include a nickname, $bnick is $null.

Remember that **$banmask** is usually a **wildcard string** which means that it will be matching wildcard strings in your remote users section. For example, if someone sets a ban of ***!k*d@*.com** it will match users:

*!khaled@mirc.com
*!kha*d@*am.d*mo?.*
!k@*

Comparing levels

You can **compare the levels** of the banner and the banned by prefixing the line with <,>,<=,=>,<>, or =, in the following way:

on >=2:BAN:#mIRC:/msg $chan $nick banned $banmask (legal)
on 1:BAN:#mIRC:/msg $chan $nick banned $banmask (illegal)

In this situation, if the banners level is larger than or equal to the banned users level, then it is a legal ban. Otherwise, it defaults to the second ON BAN line which indicates that it is an illegal ban. Remember, this is comparing the banners and banned users levels and has nothing to do with with the level 2 in the definition.

Note: These events only work on nicknames because the IRC server only sends the nickname of the user being banned/unbanned and not an address. Also, IP addresses will not be matched against named addresses, and banmasks ending in @* will be ignored since this can match almost any user address.

on CHAT/SERV

The **on CHAT** and **on SERV** events trigger when a message is sent to a dcc chat or dcc fserve window.

Format: on <level>:CHAT:<matchtext>:<commands>
Example: on 1:CHAT:*help*:/msg $nick what's the problem?

For an explanation of **matchtext** see the on TEXT event.

Examples

on 1:CHAT:boo!:/msg =$nick Boo back at ya!

This triggers when someone in a dcc chat window says boo! The equal sign in **=$nick** is required to send the reply as a dcc chat message. If no equal sign is used, the message is sent as a private irc server message.

on 1:SERV:bye:/msg =$nick Thanks for using my fileserver, bye!

This triggers when a user in a dcc fileserver session says the word bye to quit the fileserver. You can also refer to the **$cd** identifier to reference the current directory a fileserve is in.

Note: These events react to **all** users level 1 and above because of the way dcc chat works.

on CONNECT

The **on CONNECT** event triggers when mIRC connects to an IRC Server right after the MOTD is displayed.

Format: on <level>:CONNECT:<commands>
Example: on 1:CONNECT:/join #new2irc

The **on DISCONNECT** event uses exactly the same format as above and triggers when you quit or are disconnected from an IRC Server.

Examples

on 1:CONNECT:/echo Connected to $server at $time with nickname $nick

This triggers after mIRC connects to a server.

on 1:DISCONNECT:/echo Disconnected from $server at $time with nickname $nick

This triggers when mIRC is disconnected from a server.

Note: The on CONNECT event is essentially the same as the Perform section.

Ctcp Events

CTCP stands for **Client-To-Client-Protocol** which is a special type of communication between IRC Clients. By creating CTCP events, you can make your mIRC react to commands or requests from other users. CTCP events use the format:

ctcp <level>:<matchtext>:<*|#|?>:<commands>

The **level** is the access level required to access this event, the **matchtext** is the actual CTCP being sent, the ***#?** specify whether to react to any message, to channel messages, or to private messages respectively, and the **commands** are the commands that will be performed if this event triggers successfully.

Examples

The following examples should give you an idea of how to create simple CTCP events.

A Basic CTCP event

ctcp 1:help:*:/msg $nick help yourself!

The above ctcp event would react to a **/ctcp yournick help** message either in a channel or private message. Since it has access level 1, this means that any user can access it because 1 is the lowest access level.

Giving Op status to a friend

=5:*!khaled@mirc.com

ctcp 5:opme:?:/mode $2 +o $nick
ctcp 5:inviteme:?:/invite $nick $2

These definitions would allow the above level 5 user to send you the ctcp **/ctcp yournick opme #mIRC** and if you are on an Op on channel #mIRC, the above script would automatically Op him. The user can also send you the ctcp **/ctcp yournick inviteme #mIRC**, and you would invite him to channel #mIRC.

Note: By giving the user access level =5, the user is limited only to level 5 events. If I had given the user access level 5, then the user would be able to access all events which have access level 5 **and** below.

Changing a standard CTCP reply

ctcp 1:ping:?:/notice $nick Ouch! | /halt

This will react to the standard ping CTCP and will reply with "Ouch!". The /halt at the end of the line prevents the standard ping reply from being sent. If you don't use the /halt, the standard reply to PING will be sent.

ctcp 1:time:?:/notice $nick The time here is around $time | /halt

This will react to the standard time CTCP and will reply with the above message. Again, the /halt prevents the standard time reply from being sent.

Note: You can't prevent the standard version reply from being sent.

Controlling your mIRC remotely

```
100:*!khaled@mirc.com
```

```
ctcp 100:quit:?:/notice $nick Okay boss, I'm quitting... see you later! | /quit
```

The above definition shows how you can give yourself a high access level to access the quit event, and you can tell your mIRC to quit IRC from another IRC Client.

```
ctcp 100:send:?:/dcc send $nick $1-
```

This definition allows you to ask your mIRC to send you whatever file you specify to the IRC Client you're using from another location, for example by using the ctcp **/ctcp yournick send homework.txt**.

```
ctcp 100:*:?:$1-
```

This event definition allows you to execute any command remotely. So if you send a **/ctcp yournick echo Hi!**, your script will execute the command **echo Hi!**. This is a potentially dangerous event definition since if you allow anyone else to access it, they will be able to perform any command they want on your computer.

Wildcards and Variables

```
ctcp 1:*help*:#:/notice $nick I can see that you need some help
```

By using the *** and ? wildcard characters**, you can match any incoming text. So if a user sends you a ctcp which has the word help in it anywhere, the above notice will be sent.

```
ctcp 1:%password:?:/notice $nick Your access has been authorized
```

By using Variables in the matchtext section, you can change the value of %password whenever you want without having to change the event definition. So if you set %password to the value "moo" and someone sends you a "moo" ctcp, it will match %variable and the notice will the above message will be sent.

on CTCPREPLY

The **on CTCPREPLY** event triggers when a user sends a standard ctcp reply to a ctcp that you initiated.

Format:	on <level>:CTCPREPLY:<matchtext>:<commands>
Example:	on 1:CTCPREPLY:VERSION*:/echo $nick is using IRC client: $1-

For an explanation of **matchtext** see the on TEXT event.

Examples

on 1:CTCPREPLY:PING*:/echo $nick replied to my ping!

This triggers when a user replies to a ctcp PING that you sent.

on DCCSERVER

This event triggers when someone tries to connect to your DCC Server. The purpose of this event is to allow you to monitor connections and to prevent someone from connecting to your server by using /halt.

Format:	on <level>:DCCSERVER:<Chat|Send|Fserve>:<commands>
Example:	on 1:DCCSERVER:Send:echo $nick $address wants to send you $filename

Examples

```
on 1:DCCSERVER:Chat:/echo $nick $address wants to chat with you!

on 1:DCCSERVER:Send:if (.exe isin $filename) /halt
```

The above event checks triggers when someone wants to send you a file, and if the file they're sending ends in .exe, it cancels the send by using the /halt command.

on DNS

The **on DNS** event triggers when a /dns query either succeeds or fails.

```
Format:     on <level>:DNS:<commands>
Example:    on 1:DNS:/notice $me Resolved: $raddress
```

Examples

```
on 1:DNS:{
  var %n = $dns(0)
  echo 4 Found %n addresses
  while (%n > 0) {
    echo 4 dns: $dns(%n) nick: $dns(%n).nick addr: $dns(%n).addr ip: $dns(%n).ip
    dec %n
  }
}
```

Note: This event is also triggered if you try to /dns a nickname, and the nickname is not on IRC.

The $dns(N) identifier
This identifier can be used only in the on DNS event, and returns the address that was resolved and any associated IP addresses.

Properties: nick, addr, ip

$dns(N) without a property returns the address being resolved.

You can use N = 0 to return the **number** of addresses found.

on ERROR

The **on ERROR** event triggers when an IRC Server sends an ERROR message, this usually occurrs on a disconnection.

Format:	on <level>:ERROR:<matchtext>:<commands>
Example:	on 1:ERROR:*server full*:/echo Try another server!

For an explanation of **matchtext** see the on TEXT event.

Examples

on 1:ERROR:*banned*:/echo I'm banned from this server *mumble*!

This triggers when you try to connect to a server and it tells you you are banned. The **$1-** parameters refer to the full error message.

Note: This event is not related to any kind of error reporting in mIRC itself.

on EXIT

The **on EXIT** event triggers when mIRC itself is closed.

Format:	on <level>:EXIT:<commands>
Example:	on 1:EXIT:/echo exiting mIRC!

Examples

The main purpose of this command is to allow scripts to exit cleanly, unset variables, save settings, etc. when mIRC exits.

on FILESENT/FILERCVD

The **on FILESENT** and **on FILERCVD** events trigger when a dcc send or dcc get succeeds.

Format:	on <level>:FILESENT:<filename[,filename]>:<commands>
Example:	on 1:FILESENT:*.txt:/msg $nick I have successfully sent you the $filename text file

The **on SENDFAIL** and **on GETFAIL** events use the same format as above, and trigger when a dcc send or dcc get fails.

Examples

on 1:FILESENT:*.txt,*.ini:/echo Sent $filename to $nick $address

This triggers when a dcc send succeeds in sending a .txt or .ini file to a user. **$filename** refers to the filename that was transmitted.

on 1:FILERCVD:*.txt,*.ini:/echo Received $filename from $nick | /run notepad.exe $filename

This triggers when a dcc get succeeds in getting a .txt or .ini file from a user.

on 1:SENDFAIL:*.txt:/echo I failed to send the text file $filename to $nick

This triggers when a dcc send failed to send a .txt file to a user.

on 1:GETFAIL:*.zip:/echo I failed to get the zip file $filename from $nick

This triggers when a dcc get failed to get a .zip file from a user.

on HOTLINK

The **on HOTLINK** event triggers when move your mouse over a specific word in a line of text in a window.

Format: on <level>:HOTLINK:<matchtext>:<*#?=!@>:<commands>

Examples

This event works somewhat differently from other events, and is best explained with an example:

```
on ^1:HOTLINK:*help*:#:{
  if ($1 == helpme) return
  halt
}

on 1:HOTLINK:*:*:echo clicked word $1 in line $hotline
```

The first ^ event is triggered when you move your mouse over a word that matches ***help*** in a channel window. You can then check **$1** to see if you want the hotlink hand to appear over the word. If you **halt** the event, no hand will appear. This allows you to filter a word based on context.

The **$hotline** identifier returns the full line which contained the hotlink trigger.

The second non-^ event is triggered when you double-click on a word which has been filtered through the first hotlink event.

Note: The script for hotlink events should be as small and as fast as possible since the event triggers each time the mouse is moved over a word.

on INPUT

The **on INPUT** event triggers when you enter text in an editbox and press enter.

Format: on <level>:INPUT:<*#?=!@>:<commands>
Example: on 1:INPUT:#mIRC:/echo You entered the text " $1- " in the #mIRC window

Examples

on 1:INPUT:#:/echo I just mumbled " $1- " in a channel

Triggers when you enter text in an editbox in a channel window and press enter. The **$1-** parameters refer to the text that you entered. If you /halt this event, you can prevent mIRC itself from processing your message.

on 1:INPUT:?:/echo I just mumbled " $1- " in a query window
on 1:INPUT:=:/echo I just mumbled " $1- " in a dcc chat
on 1:INPUT:!:/echo I just mumbled " $1- " in a fileserver

You can also specify a specific channel/window name instead of *#?=!@.

You can use the **$ctrlenter** identifier to test whether Control+Enter was pressed when the user entered the text.

Note: You can use commands such as /say with on INPUT and they will send the message to the window inwhich you're typing, however most commands/events don't work this way and require you to specify the actual destination of a message.

on INVITE

The **on INVITE** event triggers when a user invites you to a channel.

```
Format:     on <level>:INVITE:<#[,#]>:<commands>
Example:    on 1:INVITE:#mIRC:/join $chan
```

Examples

on 2:INVITE:#:/join $chan | /timer 1 3 /describe $chan appears in a puff of smoke!

This triggers when a user with access level 2 invites you to any channel.

Note: If you want to automatically join a channel when someone invites you, it's easier to turn on the Auto-Join option in the Options dialog.

on JOIN/PART

The **on JOIN** and **on PART** events trigger when a user joins or parts a channel.

Format:	on <level>:JOIN:<#[,#]>:<commands>
Example:	on 1:JOIN:#mirc,#irchelp:/msg $nick hiya!

Examples

on 1:JOIN:#:/msg $chan Welcome $nick

This triggers when any user joins any channel which you are on.

on 5:PART:#mIRC,#newbies:/describe $chan waves bye-bye to $nick *sniff*

This triggers when a user with access level 5 leaves channels #mIRC or #newbies.

on KEYDOWN/KEYUP

The **on KEYDOWN** and **on KEYUP** events trigger when a user presses or releases a key in a custom window.

Format: on <level>:KEYDOWN:<@>:<key,...,keyN>:<commands>
Example: on 1:KEYDOWN:@:*:echo User pressed key $keyval in $active

Examples

```
on 1:KEYDOWN:@frog:32:echo user pressed space bar in @frog
```

This triggers when when a user presses the spacebar key in window @frog.

```
on 1:KEYDOWN:@:37,38,39,40:echo pressed cursor key $keyval $keyrpt
```

This triggers when a user presses any of the cursor keys in any window.

$keyval returns the key code of the key being pressed.

$keychar returns the actual letter of the key being pressed.

$keyrpt returns 1 if the key is repeating due to a user holding down the key.

on KICK

The **on KICK** event triggers when a user is kicked from a channel.

Format: on <level>:KICK:<#[,#]>:<commands>
Example: on 1:KICK:#mirc,#irchelp:/msg $nick Oops! ;)

Examples

on 5:KICK:#:/invite $knick $chan | /msg $nick Hey, $knick is my friend!

This triggers when when a user who has access level 5 is kicked out of any channel. **$knick** refers to the nickname of the user who was kicked.

Comparing levels

You can **compare the levels** of the kicker and the kicked users by prefixing the line with <,>,<=,=>,<>, or =, in the following way:

on >=2:KICK:#mIRC:/msg $chan $nick kicked $knick (legal)
on 1:KICK:#mIRC:/msg $chan $nick kicked $knick (illegal)

In this situation, if the kickers level is larger than or equal to the kicked users level, then it is a legal kick. Otherwise, it defaults to the second on KICK event which indicates that it is an illegal kick. Remember, this is comparing the **kickers and kicked users levels** and has nothing to do with with the level "2" in the definition.

Note: This event only works on a nickname because the IRC server only sends the nickname of the user being kicked and not an address.

on LOAD/START

The **on LOAD** event triggers the first time a script file is ever loaded.

Format:	on <level>:LOAD:<commands>
Example:	on 1:LOAD:/echo mIRC Script Loaded!

The **on START** event uses the same format, and triggers the first time a script is ever loaded and also every time after that when when mIRC is run.

Examples

on 1:LOAD:/echo Performing one-time initialization for this script!

Triggers the first time a script is ever loaded. The purpose of this event is to perform a one-time initialization of settings.

on 1:START:/echo Performing regular initialization for this script!

Triggers the first time a script is ever loaded, and also every time after that when scripts are loaded when mIRC is first run. The purpose of this event is to perform general initialization settings.

Note: When a file is loaded in the remote dialog, it's initialization sections are not run until after the dialog is closed. Only **one** of each of these events is allowed in a script.

on UNLOAD

The **on UNLOAD** event triggers in a script when the script is unloaded.

Format:	on <level>:UNLOAD:<commands>
Example:	on 1:UNLOAD:/echo mIRC Script unloading

Examples

```
on 1:UNLOAD:/echo Unloading script $script
```

Triggers in the script when it is unloaded. The purpose of this event is to allow a script to cleanup.

on MIDIEND/WAVEEND/MP3END

The **on MIDIEND**, **on WAVEEND**, and **on MP3END** events trigger when mIRC finishes playing a sound.

Format:	on <level>:MIDIEND:<commands>
Example:	on 1:MIDIEND:/splay jazzy.mid

Examples

on 1:WAVEEND:/echo Finished playing $filename

Triggers when a wave file finishes playing. See the Playing Sounds section for more information.

Note: These events will not trigger if you use /splay to play another sound or to stop a sound being played, they only trigger if the sound finishes playing completely.

on MODE

The **on MODE** event triggers when a user changes a channel mode.

Format:	on <level>:MODE:<#[,#]>:<commands>
Example:	on 1:MODE:#mIRC:/notice $me $nick changed $chan mode to $1-

The **on SERVERMODE** event uses the same format, and triggers when an IRC Server changes a channel mode.

Examples

on @1:MODE:#:/notice $me $nick changed $chan mode to $1-

This triggers when a user changes a mode on any channel where you have Ops (the @ at sign specifies the Op requirement, see the Access Levels section for more info). The actual parameters of the mode change are stored in **$1-** which you would need to parse yourself to enforce a particluar mode.

Note: These events only trigger on **channel** mode changes not user mode changes such as ops, bans, etc.

on NICK

The **on NICK** event triggers when a user changes nickname while on the same channel as you.

Format:	on <level>:NICK:<commands>
Example:	on 1:NICK:/echo $newnick was previously known as $nick

Examples

on 1:NICK:/describe $newnick thinks $nick was a nicer nickname!

This triggers when a user changes their nickname on a channel.

on NOSOUND

The **on NOSOUND** event triggers when a user sends a Sound Request to play a sound and you don't have that sound.

Format: on <level>:NOSOUND:<commands>
Example: on 1:NOSOUND:/notice $me Oops, $nick has $filename and I don't!

Examples

on 1:NOSOUND:/msg $nick ! $+ $nick $filename

This triggers when you don't have the requested sound. **$filename** refers to the name of the file that was requested.

Note: This will trigger whether the Warn if sound doesn't exist option is turned on or off.

on NOTIFY/UNOTIFY

The **on NOTIFY** and **on UNOTIFY** events trigger when a user in your notify list joins or leaves IRC.

Format:	on <level>:NOTIFY:<commands>
Example:	on 1:NOTIFY:/msg $nick Hiya! :)

Examples

on 1:NOTIFY:/msg $nick Hi! I'm in #mIRC_Lounge, come over!

This triggers when a user in your notify list joins IRC.

on 1:UNOTIFY:/notice $me $nick just left IRC *sniff*

This triggers when a user in your notify list leaves IRC.

on OP/DEOP

The **on OP** and **on DEOP** events trigger when a user on a channel is opped or deopped.

Format: on <level>:OP:<#[,#]>:<commands>
Example: on 1:OP:#mirc,#irchelp:/msg $nick Please don't abuse your Op status

The **on VOICE/DEVOICE** and **on HELP/DEHELP** events use the same format and trigger when a user is voiced/devoiced or helped/dehelped respectively.

The **on SERVEROP** event also uses the same format and triggers when a user is opped by a **server**.

The **on RAWMODE** event triggers independently of these events and allow you to parse the raw mode line yourself.

Examples

```
on 9:OP:#:/mode $chan -o $opnick
on 9:VOICE:#:/mode $chan -v $vnick
on 9:HELP:#:/mode $chan -h $hnick
```

This triggers when a user with access level 9 is opped/voiced/helped on any channel. **$opnick** refers to the nickname of the person being opped/deopped, **$vnick** the person being voiced/devoiced, and **$hnick** the person being helped/dehelped.

```
on 1:DEOP:#beginner:/mode $chan +o $opnick
```

This triggers when any Op is deopped on channel #beginner.

```
on 1:SERVEROP:#:/mode $chan -o $opnick
```

This triggers when an irc server ops a user on any channel. You immediately deop them.

Comparing levels

You can **compare the levels** of the opper and the opped by prefixing the line with <,>,<=,=>,<>, or =, in the following way:

```
on >=1:DEOP:#mIRC:/msg $chan $nick deopped $opnick (legal)
on 1:DEOP:#mIRC:/msg $chan $nick deopped $opnick (illegal)
```

In this situation, if the deoppers level is larger than or equal to the deopped users level, then it is a legal deop. Otherwise, it defaults to the second line which indicates that it is an illegal deop. Remember, this is comparing the oppers and opped users levels and has nothing to do with with the level 2 in the definition.

Note: These events only work on nicknames because the IRC server only sends the nickname of the user being affected and not their address.

on RAWMODE

```
on @1:RAWMODE:#:/echo $chan Raw mode line: $nick set $1-
```

The **on RAWMODE** event allows you to parse the raw mode change yourself, the raw mode text is in $1-.

You can use the $mode(N) identifier with these events to list the nicks that are being affected.

The **$modefirst** and **$modelast** identifiers return $true or $false depending on whether the event is the first or last to trigger.

on OPEN/CLOSE

The **on OPEN** and **on CLOSE** events trigger for various events relating to the opening and closing of a window of different types of windows. In the case of dcc sessions, they trigger when a dcc connection has opened or closed.

Format:	on <level>:OPEN|CLOSE:<?|@|=|!|*>:<matchtext>:<commands>
Example:	on 1:CLOSE:?:echo -s closed $target query window

For an explanation of **matchtext** see the on TEXT event.

Examples

on ^1:OPEN:?:*:if ($nick == moogoat) halt

The above open event is an example of a ^ prefix event, which allows you to halt events. In this example, if the incoming message is from a user with the nickname moogoat, the query window is prevented from opening by using the /halt command.

Note: If you halt an on ^open event triggered by an incoming private message, no on TEXT/ACTION events are triggered.

on 1:OPEN:?:*:/echo -s Just opened $target query window

The above example triggers just **after** a query window is opened.

on 1:OPEN:?:*hello*:/echo -s $nick just said hello!

This example triggers if the message which trigger on OPEN contained the word **hello**. This allows you to react to user message in on OPEN instead of waiting for on TEXT to trigger.

on 1:CLOSE:?:/echo -s you just closed $target query window

This triggers when you close a query window.

on 1:OPEN:=:*:/msg =$nick Hi! I'll be with you in a second...

This triggers when a dcc chat connection is first established. The equal sign in **=$nick** is required to send the reply as a dcc chat message. If no equal sign is used, the message is sent as a private irc server message.

on 1:CLOSE:=:/notice $me $nick just left the discussion!

This triggers when a dcc chat session ends, or when you close your chat window manually.

on 1:OPEN:!:*:/msg =$nick Welcome to my fileserver!
on 1:CLOSE:!:/echo -s $nick just ended her fileserver session

These trigger when a dcc filserver session is first established and when it is closed.

Note: DCC events react to all users level 1 and above because of the way DCCs work.

on 1:CLOSE:@:/echo -s Just closed $target custom window

The above triggers when a custom window is closed. The OPEN event does not trigger for custom windows.

Note: These events do not trigger for any other types of windows. Channel windows are handled by the on JOIN/PART events.

on PING/PONG

The **on PING** event triggers when a server sends you a PING message to see if you're still connected.

The **on PONG** even triggers when you receive a PONG reply from the server after sending it a ping.

Format:	on <level>:PING:<commands>
Example:	on 1:PING:/echo -s $nick just PING'd me!

Examples

on 1:PING:/notice $me Wake up! The server is PINGing you: $1-

This triggers when the server pings you. The **$1-** parameters hold the ping message.

on 1:PONG:echo pong reply: $1-

This triggers when the server replies to your ping.

Note: You can't use this to intercept /pings to your own nickname, this is used internally by mIRC.

on PLAYEND

The **on PLAYEND** event triggers when the /play command has finished playing a file.

Format:	on <level>:PLAYEND:<commands>
Example:	on 1:PLAYEND:/echo The play command has finished playing $filename

Note: Text files can be played ie. sent to users or channels on IRC by using the /play command.

on QUIT

The **on QUIT** event triggers when a user quits IRC while on the same channel as you.

Format:	on <level>:QUIT:<commands>
Example:	on 1:QUIT:/notice $me $nick just quit IRC with the message $1-

Examples

on 1:QUIT:/ame waves bye-bye to $nick *sniff*

This triggers when a user quits IRC while on the same channel as you. The **$1-** parameters hold the user's quit message.

Raw Events

The raw event allows you to process numeric server messages that are identified only by a **number**, and non-numeric server messages which mIRC doesn't recognize internally.

There are a large number of raw events, far too many to go into here, so it is recommended that you check out the links on the mIRC Homepage for technical information. The document you are looking for is called **RFC1459**. There is also a numerics help file available which is more up-to-date than this document.

Filtering and **handling** raw messages can be very **time-consuming** because of the large number of messages that a server can send, so you should try to make sure that your scripts process this information as quickly as possible. The format of the raw event definition is:

```
raw <numeric>:<matchtext>:<commands>
```

Examples

As a **quick** example, the following script filters out the **channels list** numeric when you use the /list command.

```
raw 322:*mirc*:/echo 5 $1-
```

The above script matches numeric 322 which is the channels /list numeric and if the line returned by this numeric has the word **mirc** in it, it is printed out in the status window.

You can process **non-numeric** server messages by specifying the name of the event:

```
raw PROP:*mirc*:/echo 5 $1-
```

Note: You can prevent most raw server messages from printing out their default text by using the /halt command.

on SNOTICE

The **on SNOTICE** event triggers when you receive a server notice.

```
Format:     on <level>:SNOTICE:<matchtext>:<commands>
Example:    on 1:SNOTICE:*client connecting*:/halt
```

For an explanation of **matchtext** see the on TEXT event.

Examples

```
on 1:SNOTICE:*hack*:/splay hack.wav
```

This triggers when a server notice contains the word hack.

Note: You can prevent the default server notice from being displayed by using /halt.

on TEXT

The **on TEXT** event triggers when you receive private and/or channel messages.

Format:	on <level>:TEXT:<matchtext>:<*><?><#[,#]>:<commands>
Example:	on 1:TEXT:*help*:#mirc,#irchelp:/msg $nick what's the problem?

The **on ACTION** and **on NOTICE** events use exactly the same format as on TEXT, and trigger on an action and on a notice event respectively.

The **match text** can be a **wildcard** string, where:

*	matches any text
&	matches any word
text	matches if text contains only this word
text*	matches if text starts with this word
*text	matches if text ends with this word
text	matches if text contains this word anywhere

The **location** where this event occurrs can be specified using:

?	for any private message
#	for any channel message
#mirc	for any messages on channel #mirc
*	for any private or channel messages

Examples

on 1:TEXT:hello*:#:/msg $chan Welcome to $chan $nick!

This listens on any channel for any line beginning with the word hello and welcomes the user who said it to the channel.

on 1:TEXT:*cookie*:#food:/describe $chan gives $nick a cookie :)

This listens on channel #food for any message containing the word cookie and gives the user who said it a cookie.

on 1:ACTION:moo:#:/msg $chan Aha, I see we have a cow among us.

This listens on any channel for an action that contains the word moo and responds accordingly.

on 1:NOTICE:*:?:/msg $nick I'm AFK, back in a moment!

This listens for any private notice and responds with the message that you're away from the keyboard.

For more flexibility, you can also use Variables in place of both the matchtext and the channel parameters.

on 1:TEXT:%matchtext:%channel:/msg $nick You just said %matchtext on channel %channel

The value of %matchtext will be matched against whatever text the user sends, and the value of %channel will be matched against the channel to which the message was sent.

Note: You can't test out these events by typing text to yourself. They can only be initiated by someone else saying something in a channel or in a private message.

on TOPIC

The **on TOPIC** event triggers when a user changes a channel topic.

Format: on <level>:TOPIC:<#[,#]>:<commands>
Example: on 1:TOPIC:#mIRC:/msg $chan Hmm, odd topic!

Examples

on 1:TOPIC:#mIRC4Dummies:/describe $chan admires $nick $+ 's new topic!

This triggers when a user changes the topic on channel #mIRC4Dummies. The **$1-** parameters hold the actual text of the new topic.

on USERMODE

The **on USERMODE** event triggers when you change your usermode.

```
Format:     on <level>:USERMODE:<commands>
Example:    on 1:USERMODE:/echo You changed your usermode to $1-
```

Examples

```
on 1:USERMODE:/echo Usermode for $nick is now $1-
```

Triggers when your usermode changes. The **$1-** parameters refer to the new usermode.

on WALLOPS

The **on WALLOPS** event triggers when you receive a wallops message.

Format: on <level>:WALLOPS:<matchtext>:<commands>
Example: on 1:WALLOPS:*warning*:/echo $nick issued warning at $time

For an explanation of **matchtext** see the on TEXT event.

Examples

on 1:WALLOPS:*oink*:/splay oink.wav

This triggers when a wallops notice contains the word oink.

Note: You can prevent the default wallops message from being displayed by using /halt.

Variables

Variables are temporary storage areas to which you can assign **values** which you can use later in your scripts.

If a variable is referred to and it doesnt exist, it returns the value **$null**. The **$null** value can be used in comparisons in if-then-else statements to control branching etc.

The following **commands** allow you to create and set the values of variables.

/set [-snzuN] <%var> [value]
This sets the value of %var to the specified value.

If you specify the **-uN** switch, %var is unset after N seconds, assuming it is not set again by another script. If you specify a zero for N, the variable is unset when the script finishes.

The **-z** switch decreases %var until it reaches zero and then unsets it.
The **-n** switch makes it treat value as plain text.

/unset [-s] <%var>
This unsets and removes the specified variables from the variables list. If you specify a variable with **wildcard** characters then all matching variables will be removed.

```
/unset %test*
```

This will **remove** all variables beginning with the word %test.

You can also set/unset dynamic variables using [] brackets:

```
vartest   {
  set %a [ $+ b ] 1
  set %a [ $+ c ] 2
  set %a [ $+ d ] 3

  echo ab = %ab
  echo ac = %ac
  echo ad = %ad

  unset %a [ $+ b ] %a [ $+ c ] %a [ $+ d ]
}
```

/unsetall
This unsets and removes all variables from the variables list.

/inc [-cszuN] <%var> [value]
This increases the value of %var by value.

If you specify the **-uN** switch, %var is increased by the value **once** and then %var is unset N seconds later, assuming it is not set again by another script.

The **-c** switch increases %var once per second.

The **-z** switch decreases %var until it reaches zero and then unsets it.

/dec [-cszuN] <%var> [value]
This decreases the value of %var by value.

If you specify the **-uN** switch, %var is decreased by the value **once** and then %var is unset N seconds later, assuming it is not set again by another script.

The **-c** switch decreases %var once per second.

The **-z** switch decreases %var until it reaches zero and then unsets it.

You can also use the **equal** sign to **assign** values to variables:

```
%i = 5
%xyzi = 3.14159
%count = $1
```

And you can perform the following **operations** on variables when using the **equal** sign:

```
%x = 5 + 1
%x = 5 - %y
%x = %x * 2
%x = %z / $2
%x = $1 % 3
%x = 2 ^ %w
```

You can only perform a **single** operation in an assignment at this time.

You can also use the **$calc()** identifier which allows you to perform complex calculations.

```
//echo 1 $calc(3.14159 * (2 ^ %x % 3) - ($ticks / (10000 + 1)))
```

For **floating point** numbers you can also use the **$round(N,D)** and **$int(N)** identifiers to handle precision of the decimal digits. The number of decimals is currently limited to 5 digits.

Local Variables

Local variables are variables that exist **only** for the **duration** of the script in which they are created and can **only** be accessed from **within** that script. They can be created with the /var command:

```
/var %x
```

This creates the local variable %x in the current routine and can only be referenced from inside this routine.

```
/var %x = hello
```

This creates a local variable %x and **assigns** it the value **hello**.

You can create **multiple** local variables by separating them with commas:

```
/var %x = hello, %y, %z = $me
```

```
loop {
  var %x = 1
  :next
  echo item %x
  inc %x
  if (%x < 10) goto next
}
```

Note: You can use /var -s to make a variable show the result when a value is set.

Identifiers

$var(%var,N)
Returns the Nth matching variable name.

Properties: value, local

You can use a wildcard in the variable name.

If N = 0, returns total number of matching variable names.

Note: This searches both local and global variables.

Identifiers

Identifiers return specific values eg. **$time** would return the **current time**. Whenever mIRC finds an identifier in a command or script, it **replaces** it with the **current value** of that identifier. Many identifers also perform functions on data that you supply and then return a result.

Identifiers which cannot be evaluated or evaluate to no value return the value **$null**. The **$null** value can be used in comparisons in if-then-else statements to control branching etc. You can also place identifiers or variables **inside** the brackets of other identifiers and they will be evaluated.

The identifiers are organized by groups as follows:

Time and Date Identifiers
File and Directory Identifiers
Nick and Address Identifiers
Text and Number Identifiers
Token Identifiers
Window Identifiers
Other Identifiers

There are also specialized identifiers for Agents, Dde Control, Custom Windows, Picture Windows, Sockets, Remote Scripts, Dialogs, Binary Files, Hash Tables, Regular Expressions, COM Objects, Signals, Multi-server, Internal Address List, and Sounds.

Time and Date Identifiers

$asctime(N,format)
Returns the time and date in text format associated with the $ctime time value.

$asctime(793947600)	returns the default text format for this time value
$asctime(hh:nn:ss)	returns the current time in this format
$asctime(793947600,dd/mm/yy)	returns the date for this time value

The identifiers $time(), $date(), and $gmt() can also be used with the format specification below.

The format parameter is **optional**, if it isn't provided, a default format is used. The format can be a combination of the following items:

Year	yy
	yyyy
Month	m
	mm
	mmm
	mmmm
Day	d
	dd
	ddd
	dddd
Hours	h
	hh
	H
	HH
Minutes	n
	nn
Seconds	s
	ss
AM/PM	t
	tt
	T
	TT
Ordinal	oo
Timezone	z
	zz
	zzz

Note: You can specify both the N and format parameters, or only one or the other.

$ctime
Returns total number of seconds elapsed since 00:00:00 GMT, January 1, 1970 based on your system time.

$ctime(text)
Returns the number of seconds elapsed since 00:00:00 GMT, January 1, 1970 based on the date that you specify.

$ctime(January 1 1970 00:00:00)
$ctime(3rd August 1987 3:46pm)

$ctime(21/4/72 1:30:37)
$ctime(Wed 1998-3-27 21:16)

$ctimer
Returns name of timer that triggered the current script.

$date
Returns the current date in day/month/year format.

For the date in US format you can use $adate.

$day
Returns the name of the current day ie. Monday, Tuesday, etc.

$daylight
Returns seconds offset if daylight savings is in effect, and 0 if not.

$duration(seconds,N)
Returns the specified number of seconds in a week/day/hour/minute/second format.

The N parameter is optional. If N = 2, the result does not include the seconds value.

Note: This identifier can also take its own output and change it back into seconds.

$fulldate
Returns the current date in the format: Wed Jun 26 21:41:02 1996

$gmt
Returns the current GMT time value in $ctime format.

$idle
Returns your current idle time (same time as that returned by a ctcp finger).

$ltimer
Returns the number of the last timer that was started by the /timer command.

$online
Returns the number of seconds elapsed in the Timer dialog.

$ticks
Returns the number of ticks since your O/S was first started.

$time
Returns the current time in hour:minute:second format.

$timer(N/name)
Returns the timer id of the **Nth** timer in the timers list. You can also specify a timer **name** instead of a number. This identifier works in conjunction with the /timer command.

Properties: com, time, reps, delay, type, secs, mmt, anysc, wid, cid, hwnd

$timer(0)	returns the number of active timers
$timer(1)	returns the timer id of the 1st timer in the list
$timer(1).com	returns the command for the 1st timer in the list
$timer(3).type	returns online/offline status for the 3rd timer in the list
$timer(3).secs	returns number of seconds left before timer is triggered

$timer(3).mmt	returns $true if timer is a multimedia timer
$timer(3).anysc	returns $true if the /timer -i switch was specified

$timestamp
Returns the current time in [xx:xx] format.

$timezone
This returns your current timezone setting in seconds. For the 16bit version the return value depends on the TZ environment variable.

$timestampfmt
Returns the timestamp format as defined in the message dialog.

$uptime(mirc | server | system, N)
Returns uptime in milliseconds for specified item.

N is optional, N = 1 returns same format as $duration(), N = 2 returns same format as $duration() but without seconds, and N = 3 returns seconds instead of milliseconds.

Text and Number Identifiers

$abs(N)
Returns the absolute value of number N.

```
$abs(5)    returns 5
$abs(-1)   returns 1
```

$and(A,B)
Returns A binary and B.

$asc(C)
Returns the ascii number of the character C.

```
$asc(A)   returns 65
$asc(*)   returns 42
```

$base(N,inbase,outbase,zeropad,precision)
Converts number N from inbase to outbase. The last two parameters are optional.

```
$base(15,10,16)       returns F
$base(1.5,10,16)      returns 1.8
$base(2,10,16,3)      returns 002
```

$biton(A,N)
Returns the A value with the Nth bit turned on.

$bitoff(A,N)
Returns the A value with the Nth bit turned off.

$bytes(N,bkmgt3)
Returns comma formatted filesize.

Properties: suf

The **bkmgt** options return the result as bytes, kilobytes, etc.

The **.suf** property appends the b, k, M, G, T suffixes to the result.

The **3** option returns a result in 3 digit format.

$calc(operations)
Returns the result of the specified operations. This identifiers allows you to perform multiple operations easily. For example:

```
$calc(3.14159 * (2 ^ %x % 3) - ($ticks / (10000 + 1)))
```

$chr(N)
Returns the character with ascii number N.

```
$chr(65)   returns A
$chr(42)   returns *
```

$cos(N), $acos(N)

Return the cosine and arccosine of N.

$count(string,substring,substring2,...,substringN)
Returns the number of times substring occurrs in string.

```
$count(hello,el)   returns 1
$count(hello,l)      returns 2
```

$encode(%var | &binvar, mubt, N)
$decode(%var | &binvar, mubt, N)
Encode or decode text in %vars or &binvars using Uuencode or Mime.

The second parameter consists of switches, where m = mime, u = uucode (default), b = &binvar, and t = text (default).

The final encoded line is made up of 60-character chunks. You can specify N if you want mIRC to return the Nth chunk. N = 0 returns the total number of chunks in the line.

If encoding/decoding a &binvar, the identifiers return the actual number of characters written to the &binvar. Note that encoding uses 33% more storage space.

The last two parameters are optional, default to uuencode, and N = 1.

$int(N)
Returns the integer part of a floating point number with no rounding.

```
$int(3.14159)   returns 3
```

$isbit(A,N)
Returns 1 if the Nth bit in number A is turned on.

$islower(text)
Returns $true if text is all lower case.

$isupper(text)
Returns $true if text is all upper case.

$left(text,N)
Returns the N left characters of text.

```
$left(goodbye,4)   returns good
```

If N is a negative value, it returns all but N characters.

$len(text)
Returns the length of text.

```
$len(#mIRC)   returns 5
```

$log(N)
Returns the natural logarithm of N.

$longip(address)
Converts an IP address into a long value and vice-versa.

```
$longip(158.152.50.239)   returns 2660774639
```

$longip(2660774639)   returns 158.152.50.239

$lower(text)
Returns text in lowercase.

$lower(HELLO)   returns hello

$mid(text,S,N)
Returns N characters starting at position S in text.

$mid(othello,3,4)   returns hell

If N is zero, it returns the number of characters from S to the end of the line.

You may also use negative numbers for S or N.

$not(A)
Returns the binary not value of A.

$or(A,B)
Returns A binary or B.

$ord(N)
Appends st, nd, rd, th as appropriate to the number N.

$pos(text,string,N)
Returns a number indicating the position of the Nth occurrence of string in text.

$pos(hello there,e,1)   returns 2
$pos(hello there,e,2)   returns 9
$pos(hello there,a,1)   returns $null

If N is zero, it returns the number of times string appears in text.

Note: You can use $poscs() for a case-sensitive version.

$rand(v1,v2)
This works in two ways. If you supply it with numbers for v1 and v2, it returns a random number between v1 and v2. If you supply it with letters, it returns a random letter between letters v1 and v2.

$rand(a,z)   returns a letter in the range a,b,c,...,z
$rand(A,Z)   returns a letter in the range A,B,C,...,Z
$rand(0,N)   returns a number in the range 0,1,2,...,N

$remove(string,substring,...)
Removes any occurrence of substring in string.

$remove(abcdefg,cd)   returns abefg

You can also specify **multiple** remove parameters:

$remove(abcdefg,a,c,e,g)   returns bdf

Note: You can use $removecs() for a case-sensitive version.

$replace(string,substring,newstring,...)
Replaces any occurrence of substring in string with newstring.

$replace(abcdefg,cd,xyz)   returns abxyzefg

You can also specify **multiple** replace parameters:

$replace(abcdefg,a,A,b,B,c,C,d,D)   returns ABCDefg

Note: You can use $replacecs() for a case-sensitive version.

$right(text,N)
Returns the N right characters of text.

$right(othello,5)   returns hello

If N is a negative value, it returns all but N characters.

$round(N,D)
Returns the specified floating point number rounded to the Dth decimal digit.

$round(3.14159,2)   returns 3.14

The D parameter is optional.

$sin(N), $asin(N)
Return the sine and arcsine of N.

$sqrt(N)
Returns the square root of N.

$str(text,N)
Returns text repeated N times.

$str(ho,3)   returns hohoho

$strip(text,burcmo)
Returns text with all bold, underline, reverse, and color control codes stripped out.

The **burcmo** parameter is optional, if used, it strips only the specified types of characters.

The **m** applies the strip settings in the Messages dialog, and the **o** applies the "only if..." settings in the messages dialog.

$stripped
Returns the **number** of control codes that were stripped from an incoming message, if any. This can be used in any script event that triggers when a message is received from a user.

$tan(N), $atan(N)
Return the tangent and arctangent of N.

$upper(text)
Returns text in uppercase.

$upper(hello)   returns HELLO

$wrap(text, font, size, width, [word,] N)
Returns Nth line in text wrapped to the specified width in pixels.

If the optional word parameter is 1, text is wrapped at whole words (which is the default).

You can specify 0 for N to return the total number of wrapped lines.

$xor(A,B)
Returns A binary xor B.

File and Directory Identifiers

$abook(nick,N)
Returns information about nicknames listed in the address book.

Properties: nick, info, email, website, picture, noteN

Allowed formats: $abook(nick) $abook(N) $abook(nick,N) where nick can also be a wildcard.

$alias(N/filename)
Returns the filename for the Nth loaded alias file. If you specify a filename, it returns $null if the file isn't loaded.

$alias(0)	return the number of alias files loaded
$alias(2)	returns the filename of the 2nd loaded alias file
$alias(moo.txt)	returns $null if the file isn't loaded, or moo.txt if it is.

$crc(filename)
Returns the CRC of the contents of the specified file.

$disk(C)
Returns information about the specified hard disk.

Properties: type, free, label, size, unc

$disk(c:)   returns $true if drive c: exists, otherwise $false

The **unc** property returns the path for a network drive.

$exists(file/dir)
Returns $true if a file or dir exists and $false if it doesn't.

$exists(c:\mirc\mirc.exe)   returns $true or $false.

$file(filename)
Returns information about the specified file.

Properties: size, ctime, mtime, atime

$file(mirc.exe).size	returns the file size
$file(mirc.exe).ctime	returns creation time
$file(mirc.exe).mtime	returns last modification time
$file(mirc.exe).atime	returns last access time

$filtered
Returns the number of lines that were filtered when using the /filter command.

$finddir(dir,wildcard,N,depth,@window | command)
Searches the specified directory and its subdirectories for the Nth directory name matching the wildcard specification and returns the full path and directory if it is found.

Properties: shortfn

$finddir(c:\,mirc*,1)   returns the first directory name beginning with "mirc"

If you specify a custom @window (with a listbox) instead of the N parameter, mIRC will fill the custom @window listbox with the results.

If you specify a command, the command is performed on every directory that is found. You can use $1- to refer to the directory name, eg.

//echo 1 $finddir($mircdir,*.*,0,echo $1-)

If you specify a depth, mIRC will only search N directories deep for matching filenames.

Note: Both the depth and @window/command parameters are optional.

$findfile(dir,wildcard,N,depth,@window | command)
Searches the specified directory and its subdirectories for the Nth filename matching the wildcard file specification and returns the full path and filename if it is found.

Properties: shortfn

$findfile(c:\mirc,*.exe,1)   returns c:\mirc\mirc.exe

If you specify a custom @window name (with a listbox) instead of the N parameter, mIRC will fill the custom @window listbox with the results.

If you specify a command, the command is performed on every filename that is found. You can use $1- to refer to the filename, eg.

//echo 1 $findfile($mircdir,*.*,0,echo $1-)

If you specify a depth, mIRC will only search N directories deep for matching filenames.

Note: Both the depth and @window/command parameters are optional.

$getdir
Returns the DCC Get directory specified in the DCC Options dialog.

$getdir(filespec)
Returns the DCC Get directory for the specified file type.

$getdir(*.txt)   returns c:\mirc\text\ (for example)

$ini(file,topic/N,item/N)
Returns the name/Nth position of the specified topic/item in an ini/text file.

$ini(mirc.ini,0)   returns total number of topics in mirc.ini
$ini(mirc.ini,1)   returns name of 1st topic in mirc.ini
$ini(mirc.ini,help) returns Nth position of topic help if it exists, or returns 0 if it doesn't exist

The item/N parameter is optional. If you specify specify N = 0, it returns the total number of topics/items.

$isdir(dirname)
Returns $true if the specified directory exists, otherwise $false.

$isfile(filename)

Returns $true if the specified file exists, otherwise $false.

$lines(filename)
Returns the total number of lines in the specified text file.

$lines(c:\irc\kicks.txt)   returns the total number of lines in c:\irc\kicks.txt

$logdir
Returns the Logs directory as specified in the Logging section of the Options dialog.

$longfn(filename)
Returns long version of a short filename, only works in 32bit mIRC. In 16bit, returns same filename.

$mididir
Returns the Midi directory specified in the Sound Requests section of the Options dialog.

$mircdir
Returns the current directory of the mIRC program.

$mircexe
Returns the full path and filename of the mIRC exe file.

$mircini
Returns the name of the main .ini file, usually mirc.ini.

$mklogfn(filename)
Returns the filename format that the logging feature uses. Appends date to filename if you have the dated logfiles option turned on in the logging dialog.

You can also use $mknickfn(nickname) to fix a nickname for use as a filename, and $mkfn(filename), which just removes invalid characters.

$nofile(filename)
Returns the path in filename without the actual filename.

$nopath(filename)
Returns filename without a path if it has one.

$nopath(c:\mirc\mirc.exe)   returns mirc.exe

$read(filename, [ntsw], [matchtext], [N])
Returns a single line of text from a file.

This identifier works in conjunction with the /write command.

//echo $read(funny.txt)

Reads a random line from the file funny.txt.

//echo $read(funny.txt, 24)

Reads line 24 from the file funny.txt.

//kick # $1 $read(kicks.txt)

Reads a random kick line from kicks.txt and uses it in the kick command.

//echo $read(info.txt, s, mirc)

Scans the file info.txt for a line beginning with the word **mirc**.

//echo $read(help.txt, w,   *help*)

Scans the file help.txt for a line matching the wildcard text *help*.

If you specify the **s** or **w** switches, you can also specify the **N** value to specify the line you wish to start searching from in the file, eg.:

//echo $read(versions.txt, w, *mirc*, 100)

If the **n** switch is specified then the line read in will not be evaluated and will be treated as plain text.

If the **first line** in the file is a **number**, it must represent the **total number of lines** in the file. If you specify N = 0, mIRC returns the value of the first line if it's a number.

If the **t** switch is specified then mIRC will treat the first line in the file as plain text, even if it is a number.

$readn
Returns the line number that was matched in a previous call to $read(). If no match was found, $readn is set to zero.

$readini(filename, [n], section, item)
Returns a single line of text from an ini file

This identifier works in conjunction with the /writeini command.

//echo $readini(mirc.ini, mIRC, nick)

Reads your nickname from the mirc.ini file.

If the **n** switch is specified then the line read in will not be evaluated and will be treated as plain text.

$sdir(dir,title)
Displays the select folder dialog and returns the selected folder. Title is optional.

$sfile(dir,title,oktext)
Displays the select file dialog and returns the selected filename. Title and oktext are optional.

//splay $sfile($wavedir,Choose a wave,Play it!)

$shortfn(filename)
Returns short version of a long filename, only works in 32bit mIRC. In 16bit, returns same filename.

Nick and Address Identifiers

$address(nickname,type)
Searches the Internal Address List for the address associated with the specified nickname.

$address(nick,1)   returns *!*user@host

If the Internal Address List doesn't contain a matching nickname, the identifier returns $null.

See **$mask()** for a list of types.

$anick
Returns your alternate nickname.

$comchan(nick,N)
Returns the names of channels which both you and nick are on.

Properties: op, help, voice,

$comchan(nick,0)	returns the total number of common channels
$comchan(nick,1)	returns the first common channel name
$comchan(nick,1).op	returns $true if you're an op on the channel

$ial(nick/mask,N)
Returns the Nth address matching mask in the IAL .

$ialchan(nick/mask,#,N)
Returns the Nth address on the specified channel matching mask in the IAL.

$ibl(#channel,N)
Returns Nth item in the Internal Ban List, or if N is 0 returns total number of items in list.

Propreties: by, date, ctime

$ibl(#mirc,1)	returns the first address in the ban list
$ibl(#mirc,1).by	returns the address of the user who set the ban
$ibl(#mirc,1).date	returns the date when the user set the ban
$ibl(#mirc,1).ctime	returns $ctime format for ban date

Note: See $chan() for more information.

$level(address)
Finds a matching address in the remote users list and returns its corresponding levels list.

$level(*!*@mirc.com)   returns =5,10,20,21,32

$link(N)
Returns the Nth item listed in the server Links window.

Properties: addr, ip, level, info

$link(0)	returns the total number of links in the links window
$link(1)	returns the Nth server address in the links window

$mask(address,type)
Returns address with a mask specified by type.

```
$mask(nick!khaled@mirc.com,1)   returns *!*khaled@mirc.com
$mask(nick!khaled@mirc.com,2)   returns *!*@mirc.com
```

The available types are:

```
0: *!user@host
1: *!*user@host
2: *!*@host
3: *!*user@*.host
4: *!*@*.host
5: nick!user@host
6: nick!*user@host
7: nick!*@host
8: nick!*user@*.host
9: nick!*@*.host
```

You can also specify a type of 10 to 19 which correspond to masks 0 to 9, but instead of using a * wildcard to replace portions of the host, mIRC uses ? wildcards to replace the numbers in the address.

This standard set of masks is also used in other identifiers and commands.

$me
Returns your current nickname.

$mnick
Returns your main nickname.

$nick(#,N/nick,aohvr,aohvr)
Returns Nth nickname in the channels nickname listbox on channel #.

Properties: color, pnick

```
$nick(#mIRC,0)   returns the the total number of nicknames on #mIRC
$nick(#mIRC,1)   returns the 1st nickname on #mIRC
```

Both **aohvr** parameters are optional. The first specifies which nicks you'd like included, and the second specifies the nicks you'd like excluded, where:

a = all nicks, o = ops, h = halfops, v = voiced, r = regular

```
$nick(#mIRC,1,o)   return the first op on #mIRC
$nick(#mIRC,0,a,o)   return the total number of nicks not including ops on #mIRC
```

The **pnick** property returns the nickname in a .@%+nick format.

Note: See the $prefix identifier for more information.

$notify(N/nick)
Returns the Nth nickname in your notify list.

Properties: ison, note, sound, whois, addr

$notify(0)	returns the number of nicknames in your notify list.
$notify(3)	returns the 3rd nickname in your notify list.
$notify(3).ison	returns $true if this user is on IRC, $false if not.
$notify(goat)	returns the Nth position of nickname goat in the notify list

$snicks
Returns a string of the currently selected nicknamess in the active channel listbox in the form:

nick1,nick2,nick3,...,nickN

$snick(#,N)
Returns the Nth selected nickname in the channel listbox on channel #.

$snick(#mIRC,0)	returns the the total number of selected nicknames on #mIRC
$snick(#mIRC,1)	returns the 1st selected nickname on #mIRC

Note: If the N parameter is not specified, it returns a line containing all selected nicknames.

$snotify
Returns the currently selected nickname in the notify list box.

Window Identifiers

$active
Returns the full name of the currently active window in mIRC.

Note: This identifier also has a multi-server counterpart.

$appactive
Returns **$true** if mIRC is the active application, otherwise it returns **$false**.

$appstate
Returns the display state of the mIRC window: minimized, maximized, normal, hidden, or tray.

$chan(N/#)
Returns information on channels that you are currently on.

Properties: topic, mode, key, limit, ial, logfile, stamp, ibl, status, inwho, wid, cid, hwnd

If you specify a number N, the Nth channel name is returned.

$chan(0)	returns the number of channels you are on
$chan(2)	returns the name of the 2nd channel you are on
$chan(2).key	returns the key of the 2nd channel you are on

$chan(4).ial returns $true if Internal Address List contains addresses of all users on this channel, $false if it doesn't.

$chan(4).inwho returns $true if you sent a /who #channel to the server to fill the IAL with addresses from that channel, and the /who is still in progress.

$chan(1).ibl returns $true if mIRC has already seen a **/mode +b** and has list of bans for the channel, or $inmode if a /mode +b is **currently** being listed. See the $ibl() identifier for more information.

If you specify a channel name, information on that channel is returned but only if you are on that channel already.

$chan(#mIRC).mode   returns the mode of channel #mIRC

The **status** property returns the value joining, joined, or kicked.

$chat(N/nick[,N])
Returns the name of the Nth open dcc chat window.

Properties: ip, status, logfile, stamp, wid, cid, hwnd

$chat(0)	returns the total number of open dcc chats
$chat(1)	returns the nickname of the 1st dcc chat window
$chat(2).ip	returns the ip address of the 2nd open dcc chat window

If you specify a nick and N, it will return info on the Nth window for that nick.

$fserv(N/nick,[N])

Returns the name of the Nth open dcc chat window.

Properties: ip, status, cd

$fserv(0)	returns the total number of open fserves
$fserv(1)	returns the nickname of the 1st fserve
$fserv(1).cd	returns the current directory of the 1st fserve

If you specify a nick and N, it will return info on the Nth window for that nick.

$get(N/nick,[N])
Returns the nickname and filename of the Nth open dcc get window.

Properties: ip, status, file, path, size, rcvd, cps, pc, secs, done, resume, wid, cid, hwnd

$get(0)	returns the total number of open dcc gets
$get(2)	returns the nickname of the 2nd dcc get
$get(2).rcvd	returns the number of bytes received for the 2nd dcc get
$get(2).cps	returns the character per second rate for the 2nd dcc get
$get(3).pc	returns the percent complete of transfer for the 3rd dcc get
$get(1).secs	returns the number seconds connection has been open
$get(1).done	returns $true if transfer was successful, $false otherwise
$get(1).resume	returns resume position if file was resumed

If you specify a nick and N, it will return info on the Nth window for that nick.

$query(N/nick)
Returns the nickname of the Nth open query window.

Properties: addr, logfile, stamp, wid, cid, hwnd

$query(0)	returns the total number of open query windows
$query(2)	returns the name of the 2nd open query window

$query(N).addr   returns the address of the Nth query, however note that this address is not available until after the user has sent you a message, and that this address **might not** be correct.

$send(N/nick[,N])
Returns the nickname and filename of the Nth open dcc send window.

Properties: ip, status, file, path, size, sent, lra, cps, pc, secs, done, resume, wid, cid, hwnd

$send(0)	returns the total number of open dcc sends
$send(2)	returns the nickname of the 2nd dcc send
$send(2).sent	returns the number of bytes sent for the 2nd dcc send
$send(2).lra	returns the last received ack for the 2nd dcc send
$send(3).pc	returns the percent complete of transfer for the 3rd dcc send
$send(3).status	returns active, inactive, or waiting for the 3rd dcc send
$send(1).secs	returns the number seconds connection has been open
$send(1).done	returns $true if transfer was succesful, $false otherwise
$send(1).resume	returns resume position if file was resumed

If you specify a nick and N, it will return info on the Nth window for that nick.

$wid
Returns window id for current script.

Token Identifiers

$addtok(text,token,C)
Adds a token to the end of text but only if it's not already in text.

$addtok(a.b.c,d,46)	returns a.b.c.d
$addtok(a.b.c.d,c,46)	returns a.b.c.d

The **C** parameter is the ascii value of the character separating the tokens.

Note: $addtokcs() is the case-sensitive version.

$deltok(text,N-N2,C)
Deletes the Nth token from text.

$deltok(a.b.c.d,3,46)	returns a.b.d
$deltok(a.b.c.d,2-3,46)	returns a.d

You can specify a negative value for N.

$findtok(text,token,N,C)
Returns the position of the Nth matching token in text.

$findtok(a.b.c.d,c,1,46)	returns 3
$findtok(a.b.c.d,e,1,46)	returns $null

If you specify **zero** for N, it returns the total number of matching tokens.

Note: $findtokcs() is the case-sensitive version.

$gettok(text,N,C)
Returns the Nth token in text.

$gettok(a.b.c.d.e,3,46)	returns c
$gettok(a.b.c.d.e,9,46)	returns $null

You can also specify a range of tokens:

$gettok(a.b.c.d.e,2-,46)	returns 2nd token onwards b.c.d.e
$gettok(a.b.c.d.e,2-4,46)	returns tokens 2 through 4 b.c.d

You can specify a negative value for N.

$instok(text,token,N,C)
Inserts token into the Nth position in text, even if it already exists in text.

$instok(a.b.d,c,3,46)	returns a.b.c.d
$instok(a.b.d,c,9,46)	returns a.b.d.c

You can specify a negative value for N.

$istok(text,token,C)
Returns $true if token exists, otherwise returns $false.

Note: $istokcs() is the case-sensitive version.

$matchtok(tokens,string,N,C)
Returns tokens that contain the specified string.

```
$matchtok(one two three, e, 0, 32) returns 2
$matchtok(one two three, e, 2, 32) returns three
```

If you specify **zero** for N, it returns the total number of matching tokens.

Note: $matchtokcs() is the case-sensitive version.

$numtok(text,C)
Returns number of tokens in text.

$puttok(text,token,N,C)
Overwrites the Nth token in text with a new token.

```
$puttok(a.b.c.d,e,2,46)   returns a.e.c.d
```

You can specify a negative value for N.

$remtok(text,token,N,C)
Removes the Nth matching token from text.

```
$remtok(a.b.c.d,b,1,46)  returns a.c.d
$remtok(a.b.c.d,e,1,46)  returns a.b.c.d
$remtok(a.c.c.d,c,1,46)  returns a.c.d
```

Note: $remtokcs() is the case-sensitive version.

$reptok(text,token,new,N,C)
Replaces the Nth matching token in text with a new token.

```
$reptok(a.b.c.d,b,e,1,46)          returns a.e.c.d
$reptok(a.b.c.d,f,e,1,46) returns a.b.c.d
$reptok(a.b.a.c,a,e,2,46)          returns a.b.e.c
```

Note: $reptokcs() is the case-sensitive version.

$sorttok(text,C,ncr)
Sorts the tokens in text.

```
$sorttok(e.d.c.b.a,46)     returns a.b.c.d.e
$sorttok(1.3.5.2.4,46,nr)returns 5.4.3.2.1
```

The default is an alphabetic sort, however you can specify n = numeric sort, c = channel nick prefix sort, r = reverse sort.

Note: $sorttokcs() is the case-sensitive version.

$wildtok(tokens,wildstring,N,C)
Returns the Nth token that matches the wildcard string.

```
$wildtok(one two three, t*, 0, 32) returns 2
$wildtok(one two three, t*e, 1, 32) returns three
```

If you specify **zero** for N, it returns the total number of matching tokens.

Note: $wildtokcs() is the case-sensitive version.

Other Identifiers

$+(n1,...,nN)
Combines all of the specified parameters, the same as using $+ in between each item.

$?*!="message"
Prompts the user for input and returns the result.

//echo $?="What is your name?"

If the user enters their name in the editbox and presses the OK button, $? will return whatever the user entered. If the user clicks on the Cancel button, $? returns nothing.

//echo $?*="What is your password?"

In this case the $?* makes any text that the user types into the editbox appear as ***** characters to prevent anyone seeing what is being entered.

//echo $?!="Shall I continue?"

In this case, a Yes/No dialog pops up. If the user clicks on Yes, $true is returned, otherwise $false is returned.

The input dialog is extended vertically to display the whole message if it is very long. You can also make text appear on different lines by using the $crlf identifier to separate the lines, eg.

//echo $?="This is on the first line. $crlf $+ And this is on the 2nd line."

Note: This identifier cannot be used in a script event. One way around this is to use a /timer to initiate an input request after the script ends.

$ansi2mirc(text)
Returns text with the ANSI codes converted into mIRC color codes.

$away
Returns the value **$true** or **$false** depending on whether you are marked as away or not.

You can also use **$awaymsg** and **$awaytime** to return your current away settings.

$bits
Returns 32 for the 32bit mIRC, or 16 for the 16bit mIRC.

$cb
Returns the first 256 characters of the clipboard contents.

$cb(N)
Returns CRLF delimited lines from text currently in the clipboard.

Properties: len

$cb(0)	returns the number of lines in the clipboard
$cb(0).len	return the total length of all lines in the clipboard
$cb(1)	returns line 1 from the clipboard

$cb(1).len	returns the length of line 1

$chantypes
mIRC supports numeric 005 token CHANTYPES, and can handle a dynamic set of channel prefixes.

$chantypes returns the list of channel prefixes which can be joined, eg. #mIRC, &mIRC.

When **not connected** to a server, mIRC uses a default $chantypes value of CHANTYPES=#&.

$chanmodes
mIRC supports numeric 005 token CHANMODES, and can handle a dynamic set of channel modes.

$chanmodes returns the list of supported channel modes, eg. '+K moo' to set the channel key to 'moo'.

When **not connected** to a server, mIRC uses a default $chanmodes value of CHANMODES=bIe,k,l.

$cmdbox
Returns $true if a command or script was initiated via the command editbox in a channel window.

$cmdline
Returns the command line that was passed to mIRC when it was first run.

$color(name/N)
Returns the Nth color index of the specified color name eg. $color(action text). If you don't specify the full name the first partial match is returned eg. $color(action)

If you specify an N value, returns the RGB value for the Nth color box.

Properties: dd

$color(action).dd	returns number in double-digit format

$cr
Returns the **carriage return** character, the same as $chr(13).

$creq
Returns current /creq settings in the DCC Options chat section dialog.

$crlf
Returns a carriagereturn/linefeed combination.

$dccignore
Returns $true if ignore types is turned on in the DCC Folders dialog, otherwise returns $false.

$dccignore(N/filename)
Returns the Nth item in the dcc ignore types editbox in the DCC Folders dialog.

If n is zero, returns number of items in list, otherwise returns Nth item in list. If a filename is specified, returns $true if it matches item in list, otherwise $false.

$dccport
Returns the port being used by the DCC Server.

$dll(name.dll, procname, data)
Returns the value resulting from a call to a DLL designed to work with mIRC.

$editbox(window,N)
Returns the text in the editbox of the specified window.

If N = 1, returns the text in the second editbox in a channel window, if it's open.

$emailaddr
Returns the email address specified in the Connect dialog.

$eval(text,N)
Evaluates the contents of text N times. If N isn't specified, the default is N = 1. If N is zero, text is not evaluated.

This allows you to recursively evaluate identifiers and variables in a line of text.

$fullname
Returns the full name specified in the Connect dialog.

$hash(text,B)
Returns a hash number based on text where B is the number of bits to use when calculating the hash number.

$highlight
Returns $true if highlighting is turned on in the Highlight dialog, otherwise returns $false.

$highlight(N/text)
Returns the Nth line in the highlight listbox, or if text is specified, returns the properties for the highlight line that matches text.

Properties: text, color, sound, flash, message, nicks.

$host
Returns your Local host name.

$iif(C,T,F)
Returns T or F depending on whether the evaluation of the Conditional C is true or false.

$iif(1 == 2, yes, no)   returns "no"

$iif() returns F if the conditional returns zero, $false, or $null. For any other value $iif() returns T.

If you don't specify the F parameter, $iif returns a T value if the condition is true, and returns nothing if it's false.

$iif(1 == 2, yes)   returns nothing

You can find out more about conditionals in the if-then-else section.

$ifmatch

Returns the first parameter of matching if-then-else comparison.

In the case of this comparison:

if (text isin sometext) { ... }

$ifmatch returns "text"

$ignore(N/address)
Returns the Nth address in the ignore list.

Properties: type, secs

```
$ignore(0)        returns the total number of addresses in the ignore list
$ignore(1)        returns the 1st address in the ignore list
$ignore(2).type returns the ignore flags for the 2nd address in the ignore list
$ignore(2).secs returns number of seconds until ignore is removed if /ignore -uN was used
```

Note: If you specify an address, the first matching address in the ignore list is returned.

$inpaste
Returns $true if a user typed **Control+V** or **Shift+Insert** to paste text into an editbox, mainly useful when processing an on INPUT event.

$input(prompt,N,title,text)
Prompts the user for input and returns the result.

The input dialog is extended vertically to display the prompt message if it is very long. You can also make text in the prompt message appear on different lines by using the $crlf identifier to separate lines.

N can be a combination of values added together:

1   - show input editbox
2   - show input password editbox
4   - ok button
8   - yesno buttons
16 - yesnocancel buttons
32 - return $ok, $yes, $no, $cancel for buttons.

By default, buttons return $true or $null, same as $?. If there is an input editbox, the ok/yes buttons always return the contents of the editbox.

64, 128, 256, and 512 show the info, question, warning, and hand icons respectively.

title is the titlebar text, and text is the default text placed in the input editbox.

N, title, and text are optional parameters.

//echo $input(Enter name:,129)

Note: This identifier cannot be used in a script event. One way around this is to use a /timer to initiate an input request after the script ends.

$ip

Returns your IP address.

$isalias(name)
Returns $true if the specified name is an alias command that exists in your aliases or scripts.

Properties: fname, alias

$isalias(join)	returns $true if you have an alias for /join
$isalias(join).fname	returns the filename in which the alias exists
$isalias(join).alias	returns the alias definition for /join

$isid
Returns $true if an alias was called as an identifier, otherwise $false.

$lf
Returns the **linefeed** character, the same as $chr(10).

$lock(item/#/N)
Returns $true or $false for the lock settings on items in the Lock dialog.

Properties: send, get, chat, fserve, run, dll, channels

You can also use $lock(N) where N returns the Nth channel in the limit channels listbox, or you can specify a channel name instead of N.

$modespl
mIRC supports numeric 005 token MODES.

$modespl returns the **maximum** number of parameters allowed per /mode, eg. if $modespl is equal to 5, you can use /mode +ooooo to set five modes at a time.

$network
Returns the name of the IRC network you are currently connected to.

Note: It may not be possible to get this info if a network doesn't provide it.

$os
Returns the version number of the operating system. The reply can be 95, 98, NT, ME, 2K, or XP.

$port
Returns the **port number** of the server to which you're currently connected.

$prefix
mIRC supports numeric 005 token PREFIX, and can handle a dynamic set of channel nickname prefixes.

$prefix returns the list of channel nickname prefixes ie. op, halfop, voice, etc. that are supported on a server.

When **not connected** to a server, mIRC uses a default $prefix value of PREFIX=(ohv)@%+.

$result
Stores the number value returned to a calling routine by the **/return** command.

$rgb(name)
Returns RGB value of specified system color name which can be one of the following: face, shadow, hilight, 3dlight, frame, and text.

$server
Returns the name of the server to which you are currently connected.

If you're not currently connected to a server, it returns $null.

$server(N/address)
Returns the address of the Nth server in your irc servers list.

Properties: desc, port, group, pass

$server(0)	returns the total number of servers in the servers list
$server(2)	returns the address of the 2nd server
$server(2).desc	returns the description of the 2nd server
$server(3).port	returns the port(s) of the 3rd server

If you specify an irc server address and it is in your servers list, it returns its associated info.

$show
Returns $false if a command is prefixed with a . to make it quiet, otherwise returns $true.

$sreq
Returns current /sreq settings in the DCC Options send section dialog.

$status
Returns server connection status.

Note: This returns **closing** during the on DISCONNECT event if the status window being closed is the cause of the disconnection.

$titlebar
Returns the text in the mIRC titlebar, set by the /titlebar command.

$url
Returns the **currently active** URL in your Web Browser.

$url(N)
Returns the Nth address in your URL list.

Properties: desc, group

$url(0)	returns the total number of items in the URL list
$url(2)	returns the address of the 2nd item in the list
$url(2).desc	returns the description of the 2nd item in the list
$url(3).group	returns the group of the 3rd item in the list

$usermode
Returns your current usermode on the irc server.

$version
Returns the version of mIRC that is being used.

Custom Windows

The **/window** command, along with a few **other** related commands listed below, allows you to **create** and **manipulate** custom windows. It's format is:

/window [-abBcCdeEfg[N]hikl[N]mnoprsvwxz] [-tN,..,N] [+bdeflLmnstx] <@name> [x y [w h]] [/command] [popup.txt] [font [size]] [iconfile [N]]

Switches and Parameters

You can specify the following switches and parameters either when first creating a window or to manipulate an existing window.

The **first** set of switches:

a = activate window
b = update horizontal scrollbar width for listbox
B = prevent window from using an internal border
c = close window
C = center window when first created
d = open as desktop window
e = single-line editbox
E = multi-line editbox
f = indicates that w h are the required width and height of the text display area as opposed to the window size
g[N] = sets/removes hilight for a window button, 0 = none, 1 = message color, 2 = hilight color
h = hide switchbar window button (window only appears in Window list)
i = dynamically associate with whatever happens to be the active connection
k[N] = hides the @ prefix in the window name, 0 = hide prefix, 1 = show prefix
l[N] = listbox, if N is specified then a side-listbox N characters wide is created
m = allow line marker to be used in window
n = minimize window
o = if opened on desktop, place ontop
p = creates a picture window
r = restore window
s = sort the main window, whether text or listbox
S = sort the side-listbox
u = remove ontop setting of a desktop window
v = close window when associated status window is closed
w= show switchbar window button
x = maximize window
z = place window button at end of switchbar

The **-t** switch used to set the **tab positions** in a listbox.

t[N,...,N] = specifies the tab positions in a listbox, if text contains tabs it will be spaced out according to these tab settings

The **third** set of switches is used to change the **appearance** of a window.

b = border
d = no border
e = 3d edge
f = dialog frame

l	= tool window
L	= tool window but window won't appear in taskbar
n	= minimize box
s	= sizable
t	= titlebar
x	= maximize box

Note: Some switches may automatically turn others on/off.

@name	window name (must prefix with a @)
x,y,w,h	left top width height
popup.txt	popup filename, loaded when needed (must be a plain non-ini text file)
/command	default command (executed whenever you enter text in an editbox)
font/size	font name and size (defaults to status window font)
iconfile/N	sets the titlebar icon for a custom window

If you want the custom window to use a menu definition in a remote script, you can specify the custom window's name as the popup filename instead of an actual popup.txt filename.

Note: If you specify -1 for any of the x,y,w,h values, a default value is used, unless the window exists in which case the current value is used.

Commands

The following commands allow you to manipulate the **lines** in a custom window.

/aline [c] <@name> <text>	add line to list
/cline [c] <@name> <N>	changes the color of the Nth line to color C
/dline [c] <@name> <N[-N2]>	delete Nth line through N2th line
/iline [c] <@name> <N> <text>	insert line at Nth line
/rline [c] <@name> <N> <text>	replace Nth line
/sline [c] <@name> <N>	select Nth line

The **c** parameter specifes a color number for the line.
The **-s** switch selects the line that was just added and clears the current selections.
The **-a** switch selects a line without clearing the current selections.
The **-h** switch highlights the window's button if it's currently minimized.
The **-p** switch forces the line of text to be wrapped if it's too long to fit on one line.
The **-r** switch is used with /cline to reset a nickname color in a channel listbox to the default color.
The **-i** switch is used with /aline and /iline to indent a newly added line.
The **-n** switch is used with /aline and /iline to prevent a line from being added if it exists.
The **-m** switch is used with /cline when coloring nicknames in a channel nicklist, and makes mIRC also color the nick in channel messages.

If you are referencing a window which uses a side-listbox, you can specify the **-l** switch in the above commands to act on the side-listbox.

/renwin <@oldname> <@newname> [topic]
This allows you to change the name of an existing window to a different one, and an optional topic can be specified.

Identifiers

The following identifiers return custom window information.

$window(N/@name)

Returns properties for a window.

Properties: x, y, w, h, dx, dy, dw, dh, bw, bh, mdi, title, state, font, fontsize, logfile, stamp, icon, ontop, type, anysc, wid, cid, hwnd

x,y,w,h	return the left, top positions, and the width and height of the window repsectively.
dx,dy	return the left, top positions of the window relative to the desktop.
dw,dh	return the width and height of the text display area.
bw,bh	return the width and height of the bitmap for a graphic window.
mdi	returns $true if the window is mdi, otherwise returns $false
state	returns minimized/maximized/hidden/normal
title	returns the text in the titlebar of the window
font	returns the name of the current font
fontsize	returns the size of the current font
fontbold	returns $true if the font is bold, otherwise returns $false
logfile	returns name of logfile if one is open for the window
stamp	returns timestamp setting
icon	returns on/off depending on whether icon is visible
ontop	returns ontop status for a window
type	returns window type
anysc	returns $true if the /window -i switch was specified
wid	returns the window id
cid	returns the associated connection id

You can also use the format **$window(@wildcard,N)** which returns the Nth window matching the wildcard window name.

Note: You can also get the .w and .h properties by specifying: -1 for the width and height of the screen, -2 for the main mIRC window, and -3 for the MDI window where all other windows inside mIRC are displayed.

$line(@name,N,T)
Returns the Nth line of text in the specified window. If N is zero, it returns the total number of lines in the window.

If you are referencing a window which uses a side-listbox, you can set T to zero to reference the display area, or set T to one to reference the side-listbox.

$fline(@,wildtext,N,T)
Returns Nth position of line which matches the specified wildcard text. If T is not zero, it acts on the side-listbox.

$sline(@name,N)
Returns the Nth selected line in a listbox (only works in listboxes). If N is zero, it returns the total number of selected lines in the window.

Properties: ln

$sline(@name,N).ln	returns the **line number** of the Nth selected line

Examples

/window @test

This would create a window named @test with all of the default settings.

```
/window -els @clones 10 10 clones.txt
```

This would create a window with a sorted listbox and an editbox, positioned near the top left of the mIRC window, and using the popup file clones.txt which would appear whenever you click the right mouse button in the listbox.

Note: The popup text file must be plain text in a non-ini format using a non-ini extension.

Picture Windows

The **picture window** is a special type of custom window that can display a combination of text, graphics, and pictures, and can trigger script events relating to mouse clicks and movements.

Once you have opened a picture window using the /window command, you can use the following commands and identifiers to draw and monitor activity in this window.

Drawing commands

/drawdot -ihnr @ <color> <size> <x y> [<x y>...]
Draws a dot using the specified color and size at the x,y co-ordinates. Multiple co-ordinates can be provided.

The **-i** switch draws in inverse mode.

The **-h** switch highlites the windows icon if it's minimized.

The **-n** switch prevents the display from being updated immediately. This allows you to make changes to the window in the background and then display the results only when you've finished. You can update the display by using any of the /draw commands with only the window name specified.

The **-r** switch indicates that the color is in RGB format. You can use $rgb(N,N,N) to create an RGB value.

The whole window can be cleared by using the /clear command, eg. /clear @name. You can also specify the **-n** switch in /clear to delay the effect as described above.

/drawline -ihnr @ <color> <size> <x y> <x y> [<x y>...]
Draws a line from the first <x y> co-ordinate to the second, if more co-ordinates are specified, the line is continued. The switches are the same as those in /drawdot.

/drawrect -ihnrfecd @ <color> <size> <x y w h> [<x y w h>...]
Draws a rectangle at the specified co-ordinates of the specified width and height. If more co-ordinates are specified these are also drawn as separate rectangles.

The **-f** switch fills the rectangle with the current color.

The **-e** switch draws an Ellipse instead of a rectangle. You can draw a filled ellipse if you also specify the **-f** switch.

The **-c** switch draws a focus rectangle.

The **-d** switch draws a rounded rectangle, using the format /drawrect -d x y w h [w h], where **w** and **h** are the width and height of the ellipse used to draw the corners.

The remaining switches are the same as those in /drawdot.

/drawfill -ihnrs @ <color> <color> <x y> [filename] [<x y>...]
Fills an area with the specified color starting at the specified co-ordinates.

The **-s** switches indicates that the second color parameter is the color that should be filled (surface fill). If no **-s** is specified, the second color is the border color at which filling should stop (border fill).

The optional **[filename]** specifies a bitmap .BMP file that is 8 by 8 pixels in size and is used as a fill pattern.

The remaining switches are the same as those in /drawdot.

/drawtext -hnrpboc @ <color> [color] [fontname fontsize] <x y [w h]> <text>
Draws text at the specified co-ordinates.

The **-p** switch processes and interprets color/bold/etc. codes in the text.

The **-b** switch indicates that you have specified the second color parameter as the background color for the text.

The **-o** switch means the specified font should be in bold.

The **-c** switch means that you have specified the [w h] values as the rectangle in which text should be printed. Text will be clipped if it extends beyond this rectangle.

The remaining switches are the same as those in /drawdot.

Note: If you use a negative number for the font size, it will match the size of fonts in the font dialog.

/drawsave -bN @ <filename>
This saves the background picture of the specified picture @window to a .bmp filename.

The **-bN** switch allows you to specify the bit depth of the saved file, which can be 1, 4, 8, 16, 24, or 32.

/drawscroll -hn @ <x> <y> <x y w h> [<x> <y> <x y w h>...]
Scrolls the region inside the specified rectangle. The first <x> and <y> parameters represent the distance to scroll and can be positive or negative.

The remaining switches are the same as those in /drawdot.

/drawcopy -ihmnt @ [color] <x y w h> @ <x y [w h]>
Copies part of a picture to a different position in the same window or to another window. If the second [w h] parameters are specified, the picture is stretched/squeezed to fit.

The **-t** switch indicates that you have specified the [color] RGB value as a transparent color in the source bitmap.

The **-m** switch changes the stretch mode quality when the picture is resized.

The remaining switches are the same as those in /drawdot.

/drawpic -ihmntscl @ [color] <x y [w h]> [x y w h] <filename>
Loads and draws a Bitmap .BMP picture at the specified co-ordinates. If the first [w h] are specified, it is stretched or squeezed to fit. The second [x y w h] rectangle specifies which portion of the loaded bitmap should be displayed ie. a bitmap could contain multiple pictures.

The **-t** switch indicates that you have specified the [color] RGB value as a transparent color in the specified bitmap.

The **-s** switch indicates that you have specified the first [w h] parameters (as explained above) to squeeze/stretch the bitmap.

The **-c** switch indicates that the bitmap should be **cached**. This greatly speeds up subsequent references to this bitmap. If you specify **-c** and the bitmap is already in the cache, it is loaded and used from the cache, otherwise it is reloaded from the file. You can clear the entire cache with /drawpic -c.

The **-l** switch tiles the picture.

The **-m** switch changes the stretch mode quality when the picture is resized.

If you try to load and cache a bitmap and there are already 30 bitmaps in the cache, the bitmap with the lowest reference count is freed and replaced by the new bitmap.

If you try to load a bitmap and there isn't enough memory, mIRC repeatedly frees the least referenced bitmap and tries to load again.

The remaining switches are the same as those in /drawdot.

/drawrot -hmnbfc @ [color] <angle> [x y w h]
Rotates an area of a bitmap by the specified angle.

The **-b** switch indicates that you've specified the background color value.

The **-f** switch fits the newly rotated bitmap into the original width/height.

The **-c** switch centers the rotated image if -f isn't specified.

The **-m** switch changes the stretch mode quality when the picture is resized.

The remaining switches are the same as those in /drawdot.

/drawreplace -nr @ <color1> <color2> [x y w h]
Replaces color1 with color2 in the specified picture window.

The remaining switches are the same as those in /drawdot.

Events and Identifiers

Mouse events can be defined in a scripts menu definition.

```
menu @test {
  mouse:/echo mouse moved at $mouse.x $mouse.y in $active $1
  sclick:/echo single click at $mouse.x $mouse.y
  dclick:/echo double click at $mouse.x $mouse.y
  uclick:/echo mouse released at $mouse.x $mouse.y
  rclick:/echo single right-click at $mouse.x $mouse.y in $active $1
  lbclick:/echo mouse selected $active $1
  leave:/echo mouse left $leftwin
  drop:/echo drag and drop at $mouse.x $mouse.y
}
```

The **mouse** event is triggered when you move a mouse inside a picture window. You can use the **$mouse** identifier (see below) for the x and y position of the mouse.

The **sclick** event is triggered when you click once inside a picture window. It will also trigger if you double-click.

The **dclick** event is triggered when you double-click inside a picture window.

The **uclick** event is triggered when a mouse button is released inside a picture window.

The **lbclick** event is triggered when an item is selected in a listbox, either with the mouse or the cursor keys.

The **leave** event is triggered when the mouse is moved outside the borders of a custom window.

The **drop** event is triggered if you clicked in the window, held the button down, moved the mouse, and then let go of the button.

$mouse
Returns the x, y position of the current mouse event, and whether the left mouse button, shift key, and/or control key are pressed.

Properties: win, x, y, mx, my, dx, dy, key, lb

The $mouse identifier can be used in the mouse/click events. For $mouse.key you can use the **&** bitwise operator to check if the left button, shift key, and/or control key are pressed:

```
if ($mouse.key & 1) echo left button is pressed.
if ($mouse.key & 2) echo control key is pressed.
if ($mouse.key & 4) echo shift key is pressed.
```

The following properties can also be used:

The **.win** property returns the name of the active window.

The **.x/.y**, **.mx/.my**, and **.dx/.dy** properties return the x and y position of the mouse relative to the active window, the main mIRC window, and the desktop respectively.

The **.lb** property returns $true if a mouse event occurred over a listbox, or $false if it didn't.

$click(@,N)
This stores a history of x,y clicks for a window.

Properties: x, y

You can use /clear -c @name to clear the history of clicks for a window. If you use $click() with no properties it returns x y.

$inrect(x,y,x,y,w,h)
Returns $true if the point x y is inside the specified rectangle, and $false if it isn't.

$inpoly(x,y,a1,a2,b1,b2,...)
Returns $true if the point x y is inside the polygon defined by the specified points, and

$false if it isn't.

$onpoly(n1,n2,x,y,x,y,...)
Returns $true if two polygons overlap. n1 is the number of points in the first polygon, n2 in the second polygon.

$rgb(N,N,N)
Returns an RGB color value for use in /draw commands. If you use only **one** parameter, it assumes it is an actual RGB color value and returns N,N,N.

$getdot(@,x,y)
Returns the RGB color value of the dot at the specified position.

$height(text,font,size)
Returns height of text in pixels for the specified font.

Note: If you use a negative number for the font size, it will match the size of fonts in the font dialog.

$window
Returns the name of the window in the **leave** menu event which was just left.

$pic(filename)
Returns the size, width, and height of a bmp, jpg, or png file.

Properties: size, width, height

$width(text,font,size,B,C)
Returns width of text in pixels for the specified font.

If **B** is non-zero, the font is bold, if **C** is non-zero, control codes are processed. Both B and C are optional.

Note: If you use a negative number for the font size, it will match the size of fonts in the font dialog.

Regular Expressions

Regular expressions are basically a grammar that can be used to perfom complicated pattern matching operations. You should already know how to use Regular expressions before using the identifiers below.

It is beyond the scope of this help file to explain how Regular Expressions work. There are many websites on the internet that introduce regular expressions and provide examples.

$regex([name], text, re)
Returns N, the number of strings in text that matched the regular expression.

You can assign a **name** to a $regex() call which you can use later in $regml() to reference a match list.

If you don't specify a **name**, all identifiers use a default name which is overwritten with each call to $regex().

$regex() remembers the results for the last **ten** $regex() calls. Each time you do a match with $regex(), and you specify a name, that name's previous results are overwritten with the new results.

$regml([name], N)
This can be used to reference the back referenced (items enclosed in parentheses) values returned by a call to $regex() or $regsub().

Properties: pos

If N = 0, returns total number of match strings.

The **pos** property returns a strings position in the original match text.

$regsub([name], text, re, subtext, %var)
Performs a regular expression match, like $regex(), and then performs a substitution using subtext.

Returns N, the number of substitutions made, and assigns the result to %var.

Binary Files

mIRC allows you to read and write **binary files**, and to modify binary variables, by using the following commands and identifiers.

/bread <filename> <S> <N> <&binvar>
This reads N bytes starting at the Sth byte position in the file and stores the result in the binary variable &binvar.

/bwrite <filename> <S> [N] <text|%var|&binvar>
This writes N bytes from the specified text, %var, or &binvar, to the file starting a the Sth byte position. Any existing information at this position in the file is overwritten.

Note: If S is -1, the bytes are appended to the end of the file. If N is -1, all of the specified data is written to the file.

/bset [-t] <&binvar> <N> <asciivalue> [asciivalue ... asciivalue]
This sets the Nth byte in binary variable &binvar to the specified ascii value.

If you try to /bset a variable that doesnt exist, it is created and zero filled up to N bytes. If &binvar exists and you specify an N position beyond it's current size, it is extended to N bytes.

If you specify **multiple** asciivalues, they are copied to successive positions after byte position N.

The **-t** switch indicates that /bset should treat the values as plain text and copy them directly into &binvar.

/bunset <&binvar> [&binvar ... &binvar]
This unsets the specified list of &binvars.

/bcopy [-zc] <&binvar> <N> <&binvar> <S> <M>
This copies M bytes from position S in the second &binvar to the first &binvar at position N. This can also be used to copy overlapping parts of a &binvar to itself.

If you specify the **-z** switch, the bytes in the second &binvar that were copied are zero-filled after the copy.

If you specify the **-c** switch, the first &binvar is chopped to length N + M.

Note: If M is -1, all of the bytes from position S onwards are copied.

/breplace <&binvar> <oldvalue> <newvalue> [oldvalue newvalue...]
This replaces all matching ascii values in &binvar with new values.

/btrunc <filename> <bytes>
This truncates/extends a file to the specified length.

$bvar(&binvar,N,M)
Returns M ascii values starting from the Nth byte

Properties: text, word, nword, long, nlong

$bvar(&v,0)	returns the length of the binary variable
$bvar(&v,1)	returns ascii value at positon N
$bvar(&v,5,3)	returns ascii values from 5 to 8
$bvar(&v,5,3).text	returns plain text from 5 to 8 up to the first zero character

The word, nword, long, and nlong properties return values in host or network byte order.

Notes on &binvar binary variables
Binary variables have a maximum size of 8192 bytes, can **only** be accessed by commands /bread and /bwrite, so they can't be printed or assigned or edited, and are automatically destroyed when a script finishes processing.

$bfind(&binvar, N, M)
Searches a &binvar for a matching value, starting from position N. M can be a character value, ie. 0 to 255, or text. The search is case-insensitive.

Properties: text

$bfind(&test, 1, mirc)	finds "mirc" starting from pos 1
$bfind(&test, 5, 32)	finds char 32 (a space) from pos 5
$bfind(&test, 1, 87 65 86)	finds WAV from pos 1

You can use $bfind().text to force a text search if the search text is a number.

DLL Support

The **/dll** and **$dll()** features allow you to make calls to **DLLs** designed to work with mIRC. The **main** reason you'd want to do this is that processing information in a DLL can be far **faster** than doing so in a script, so for **intensive** data processing a DLL would be more efficient.

Note that mIRC also supports calling COM objects, for calling non-standard DLLs.

Warning: do not use DLLs from sources you do not trust. See the Accepting Files section for information on the dangers of accepting and using files from the internet.

```
   /dll <filename> <procname> [data]
   $dll(filename, procname, data)
```

Both of these open a DLL, call the procname routine, and send it the specified data. The only difference is that $dll() can return a value, like all other identifiers.

Technical notes

This section contains **technical** information for **programmers** who want to create DLLs for use with mIRC.

The **routine** in the DLL being called must be of the form:

```
int __stdcall procname(HWND mWnd, HWND aWnd, char *data, char *parms, BOOL show,
BOOL nopause)
```

mWnd is the handle to the main mIRC window.

aWnd is the handle of the window in which the command is being issued, this might not be the currently active window if the command is being called by a remote script.

data is the information that you wish to send to the DLL. On return, the DLL can fill this variable with the command it wants mIRC to perform if any.

parms is filled by the DLL on return with parameters that it wants mIRC to use when performing the command that it returns in the data variable.

Note: The data and parms variables can each hold 900 chars maximum.

show is FALSE if the . prefix was specified to make the command quiet, or TRUE otherwise.

nopause is TRUE if mIRC is in a critical routine and the DLL must not do anything that pauses processing in mIRC, eg. the DLL should not pop up a dialog.

The DLL can return an **integer** to indicate what it wants mIRC to do:

0 means that mIRC should /halt processing

1 means that mIRC should continue processing

2 means that it has filled the data variable with a command which it wants mIRC to perform, and has filled parms with the parameters to use, if any, when performing the command.

3 means that the DLL has filled the data variable with the result that $dll() as an identifier should return.

Note: You may need to create a .def file with the procedure names exported when compiling your DLL.

Keeping a DLL Loaded after a call

By default a DLL is unloaded immediately after you make the /dll or $dll() call. You can keep a DLL loaded by including a **LoadDll()** routine in your DLL, which mIRC calls the first time you load the DLL:

```
void __stdcall (*LoadDll)(LOADINFO*);

typedef struct {
  DWORD   mVersion;
  HWND    mHwnd;
  BOOL    mKeep;
} LOADINFO;
```

mVersion contains the mIRC version number in the low and high words.

mHwnd contains the window handle to the main mIRC window.

mKeep is set to TRUE by default, indicating that mIRC will keep the DLL loaded after the call. You can set mKeep to FALSE to make mIRC unload the DLL after the call (which is how previous mIRCs worked).

Unloading a DLL

You can **unload** a loaded DLL by using the **-u** switch:

```
/dll -u <filename>
```

You can browse the list of loaded DLLs by using:

```
$dll(N/filename)    returns the Nth loaded DLL
```

mIRC will **automatically** unload a DLL if it is not used for ten minutes, or when mIRC exits.

You can also define an **UnloadDll()** routine in your DLL which mIRC will call when unloading a DLL to allow it to clean up.

```
int __stdcall (*UnloadDll)(int mTimeout);
```

The **mTimeout** value can be:

0   UnloadDll() is being called due to a DLL being unloaded when mIRC exits, or unloaded with /dll -u.

1   UnloadDll() is being called due to a DLL not being used for ten minutes. The UnloadDll() routine can return 0 to keep the DLL loaded, or 1 to allow it to be unloaded.

Sockets

Socket support allows you to create your own raw socket connections in order to send and receive information. You should already be an **expert** at writing Aliases, Popups, and Scripts before attempting to use sockets.

Sockets are a **limited resource** so it is important that you understand how these commands work before trying to use them. Sockets should always be closed after they have been used to make them available to other applications.

Socket Identifiers

$sock(name,N)
This returns information about a socket connection that you created using the socket commands.

Properties: name, ip, port, status, sent, rcvd, sq, rq, ls, lr, mark, type, saddr, sport, to, wserr, wsmsg, bindip, bindport

.name is the name you give to a connection to identify it
.sent and **.rcvd** return the number of bytes sent and rcvd over that connection so far
.sq and **.rq** return the number of bytes queued in the send and receive buffers respectively
.ls and **.lr** return the number of seconds since the connection last sent and last received info
.mark is a user storage area max. 512 bytes (see /sockmark)
.type returns the socket type, TCP or UDP
.saddr and **.sport** return the source address and port of the last received UDP packet
.to returns the number of seconds the socket has been open
.wserr returns the last winsock error number that occurred on a socket
.wsmsg returns the last winsock error message match the error number

Note: You can use a wildcard name to quickly reference matching entries. The N parameter is optional, if it isn't specified it is assumed to be 1.

$sockname
$sockname is the name given to a connection to identify it. This identifier can be used in events to know which connection an event is related to.

$sockerr
$sockerr is set to a value after each socket command/event and **must** be checked after each socket command and before processing an event to see if an error occurred.

$sockbr
$sockbr is set to the number of bytes read by a /sockread command. It is used to test whether any information was in fact read from the buffer (see below for more info).

$portfree(N)
Returns $true if the specified port number is not in use, otherwise returns $false.

Socket Commands and Events

The following information lists associated commands and script events together for easy reference.

Listening for and Accepting incoming connections

/socklisten [-d] [bindip] <name> [port]
The /socklisten command listens on the specified port for connections to that port. If a port isn't specified, the port is selected randomly from the range specified in the DCC Options dcc ports section.

The **-d** switch indicates that you specified an ip address as the bind address.

on 1:socklisten:name:commands
The socklisten event is triggered when someone tries to connect to a port that you are listening on. If you want to accept the connection you **must** do it in this event using the /sockaccept command, otherwise the connection is closed.

/sockaccept <name>
The /sockaccept command accepts the current connection to your listening port and assigns it a name to identify it.

/sockrename <name> <newname>
The /sockrename command assigns a new name to an existing connection.

Opening and Closing connections

/sockopen [-d] [bindip] <name> <address> <port>
The /sockopen command initiates a connection to the specified address and port. You can specify either an ip address or a named address (which will be resolved to an ip address).

The **-d** switch means that you specified an ip address as the bind address.

on 1:sockopen:name:commands
The sockopen event is triggered when a /sockopen command is successful and a connection has been made.

/sockclose <name>
The /sockclose command closes the connection with the specified name. If you specify a wildcard name, all connections that match the wildcard are closed.

on 1:sockclose:name:commands
The sockclose event is triggered when a connection is closed by the remote connection (not you).

Sending information

/sockwrite [-tnb] <name> [numbytes] <text|%var|&binvar>
The /sockwrite command queues info to be sent on the specified connection. mIRC will then try to send that info as quickly as it can. Once it has finished sending the info, it triggers the on sockwrite event so you can send more info if you need to.

If you specify the **-t** switch, it forces mIRC to send anything beginning with a & as normal text instead of interpreting it as a binary variable. The **-n** switch appends a CRLF to the line being sent if it's not a &binvar and if it doesn't already have a CRLF.

The **-b** switch indicates that you are specifying the numbytes value which is the number of bytes you want sent.

Note: You can use a wildcard name to send the same information at once to all connections that match the wildcard.

On error: if a /sockwrite fails, it sets $sock().wserr to the error value, and triggers the on sockwrite event with $sockerr set.

on 1:sockwrite:name:commands
The sockwrite event is triggered when mIRC has finished sending all of the info which you previously queued for sending.

Note: If you try to /sockwrite while there is still info queued in the send buffer, your new info will just be added to the end of the queue up to a maximum of 8192 bytes. Any attempt to queue more than that will result in an error message, so you should check how much info is currently queued by using $sock().sq (send queue) before trying to queue info on a socket.

Reading information

on 1:sockread:name:commands
The sockread event is triggered when there is info waiting to be read on the specified connection. You can read this info using the /sockread command.

Note: If this event triggers but no /sockread is performed to attempt to read the buffer, it is assumed that no script exists that is handling this buffer, so it is cleared and the info it contained is lost.

/sockread [-fn] [numbytes] <%var|&binvar>
The /sockread command reads bytes from the receive buffer into the specified variable.

If you specify a %var variable, a line of text ending with a Carriage Return/LineFeed is read into %var. The CRLF are stripped off (this may result in %var being $null if the line only consisted of CRLF).

If you specify a &binvar then [numbytes] of info is read into the binary variable. If no [numbytes] is specified it defaults to 4096 bytes.

If you specify the **-f** switch with a %var variable, this forces mIRC to fill the %var variable with whatever text is in the receive buffer, even if it doesn't end in a CRLF.

The **-n** switch allows you to read a CRLF terminated line into a &binvar. If the incoming line does not contain a CRLF, no bytes will be read into &binvar, unless you specify the -f switch, which forces the read (same as when reading into %vars).

Note: A single /sockread may not be enough to read the entire buffer. You should keep reading until $sockbr (bytes read) is set to zero. This is far faster than letting mIRC re-trigger the event. If your script doesn't read the whole buffer, the on sockread event is re-triggered if:
   a) you were reading into a &binvar.
   b) you were reading into a %var and there is still a CRLF terminated line in the buffer waiting to be read.

Example:
This example shows you how you should process a sockread event. The socket has already been opened and has received information, so the sockread event is triggered. The socket name is **testing**. There is an explanation of each step below the sample script.

```
on 1:sockread:testing:{
  if ($sockerr > 0) return
  :nextread
  sockread %temp
  if ($sockbr == 0) return
  if (%temp == $null) %temp = -
  echo 4 %temp
  goto nextread
}
```

If **$sockerr** is larger than zero then there is a socket error. mIRC will automatically close the socket, so all you have to do is **return** from the event.

sockread %temp reads a CRLF terminated line of text and stores it in %temp. If the buffer did not contain a CRLF terminated line, %temp is not filled with anything, and $sockbr returns zero, so you should just return from the event without further processing.

If **%temp is $null** then that means the line consisted only of a CRLF which mIRC has automatically stripped out of the line, so only an empty line remains. In this case, I'm setting %temp to a dash to represent an empty line but you can do whatever you wish here.

I then **echo** the final line that was read to the status window.

Finally, I use **goto** to jump back and to continue reading remaining lines in the socket's receive buffer.

Marking a socket

/sockmark <name> [text]
The /sockmark command fills the .mark attribute of a socket with the specified info for later reference via the $sock().mark property. If you do not specify any text, the mark is cleared. The mark can hold up to 512 bytes.

Note: You can use a wildcard name to set the same information at once for all connections that match the wildcard.

Listing open sockets

/socklist [-tul] [name]
The /socklist command lists all open sockets, or if you specify the -tul switches, it lists tcp, udp, and listening sockets respectively. You can also specify a socket name or wildcard.

UDP Sockets

UDP is a connection-less protocol, ie. you can send information via UDP to other UDP addresses without needing to connect to them first.

UDP does **not** guarantee that any information you send will actually reach it's destination, ie. it isn't a reliable protocol. Also, be aware that UDP packets may not arrive in the same order as that in which they were sent. This means that you **must** perform your **own** checking to confirm that any data you sent actually reached it's destination properly.

/sockudp [-bntkd] [bindip] <name> [port] [<ipaddress> <port> [numbytes] [text|%var|&binvar]

If you specify the **-t** switch, it forces mIRC to send anything beginning with a & as normal text instead of interpreting it as a binary variable. The **-n** switch appends a CRLF to the line being sent if it's not a &binvar and if it doesn't already have a CRLF.

The **-b** switch indicates that you are specifying the numbytes value which is the number of bytes you want sent.

The **-k** switch forces the socket to stay open, this allows it to listen for data that is sent to its port via UDP. If **-k** is not specified, the socket is opened, the information is sent to the specified ipaddress and port, and the socket is then closed.

The **-d** switch means that you specified an ip address as the bind address.

If you specify a socket **name** that doesn't exist, it is created. If it does exist, the existing socket is used to send the info.

You can also specify the local **port** number that you wish to use, if it isn't specified, mIRC chooses one randomly.

ipaddress and **port** specify the destination address to which you want to send information. You can only use an IP address here.

On error: if a /sockudp fails, it sets $sock().wserr to the error value, and triggers the on sockwrite event with $sockerr set.

on 1:udpread:name:commands
The udpread event is triggered when there is info waiting to be read on a UDP socket. You can read this info using the /sockread command.

Note: If this event triggers but no /sockread is performed to attempt to read the buffer, it is assumed that no script exists that is handling this buffer, so it is cleared and the info it contained is lost.

Signals

Signals are simple way of triggering signal events in multiple scripts at the same time.

/signal [-n] <name> [parameters]
The signal command allows you to trigger signal events in all scripts that listen for signals.

By default the signal is triggered after all current scripts have finished executing. You can however use **-n** to make the script trigger immediately.

on *:SIGNAL:name:command
The on signal event triggers if a script has used the /signal command to send a signal to all scripts.

The signal **name** can contain wildcards.

The **$signal** identifier returns the signal name that caused the signal event to trigger.

The **$1-** identifier returns the parameters that were specified in the /signal command.

Note: The script that called /signal is triggered first, and then all other scripts are triggered.

Playing Sounds

mIRC supports the playing of various types of sounds with the following commands and identifiers.

/splay -cwmpq [filename | stop | pause | resume | seek | skip] [pos]
Plays the specified sound, which can be a .wav, .mid, or .mp3 file.

Where switch w = wave, m = midi, p = mp3, and q = queue for playing.

If you don't specify a folder, the sound folders in the Sound Requests dialog are searched.

You can use **stop** to stop a currently playing sound, eg. /splay stop, or /splay -w stop, to stop just wave files.

You can use **pause**, **resume**, and **seek** with mp3s.

If you specify the **pos** value when playing an mp3, eg. /splay llama.mp3 1000, mIRC will play the mp3 starting at that position.

You can also **seek** to a position in an mp3 while it's being played with eg. /splay seek 1000

To **skip** the currently playing sound, you can use /splay -wmp skip

The **-q** switch allows you to queue sounds for playing after the current sound ends.

The **-c** switch allows you to clear the play queue except for the currently playing sound.

When a sound finishes playing, it triggers a sound event.

/vol -wmpvuN [volume]
Sets the volume on your system for waves, midis, and mp3s (same as waves)

If you use the **-v** switch, this sets the master volume on your system.

The **volume** value can range from 0 to 65535.

The **-uN** switch sets the mute setting, where N = 1 is on, N = 2 is off.

$vol(wave | midi | song | master)
Gets the current volume for the specified sound setting

Properties: mute

$vol(wave).mute	returns the mute setting for wave sounds

$inwave, $inmidi, $insong
Return $true if the specified sound type is playing, $false otherwise.

Properties: fname, pos, length

$insong.fname	returns the filename of the currently playing song

$sound(type)

Returns the directory specified in the Sound Requests section of the Options dialog, where **type** can be wave, midi, mp3, wma, or ogg.

$sound(filename)
Returns either the directory for that file type, as above, or information about the sound file. Currently, only **mp3** files are supported with the following properties.

Properties: album, title, artist, year, comment, genre, track, length, version, bitrate, vbr, sample, mode, copyright, private, crc

Dialogs

mIRC allows you to create custom **Dialogs** which can be used to request input from a user, or to perform many other useful tasks.

There are two types of dialog; a **modeless** dialog, created with the **/dialog** command, and a **modal** dialog, created with the **$dialog()** identifier. Both are described below.

The /dialog command

The /dialog command allows you to create a **modeless** dialog with the **-m** switch. This type of dialog doesn't halt or return a value to the calling script, and the dialog can be displayed indefinitely and used for many purposes.

```
/dialog -mdtsonkcvie name [table] [x y w h] [text]
```

-m	create a modeless dialog using 'table' /dialog -m name table
-a	Used with -m, uses currently active window as the parent
-x	Closes a dialog without triggering any events
-d	open dialog on the desktop, used with -m
-t	set dialog title /dialog -t name text
-s	set dialog size/pos /dialog -s name x y w h
-r	centers dialog
-bp	interprets size as dbu or pixels
-o	set dialog ontop of all windows
-n	unset ontop setting
-k	click ok button
-c	click cancel button
-v	makes the dialog the active window
-ie	minimize/restore the dialog if created on the desktop

Where **name** is the name by which you'll refer to the dialog, and **table** is the dialog table name used to create dialog (see below).

The $dialog() identifier

Dialogs created with $dialog() are modal, ie. they halt the script until the dialog is closed, they return a value, and they don't allow access to other windows while the dialog is open. These types of dialogs should only be displayed for immediate user input. The $dialog() identifier works the same way as other identifiers:

```
%result = $dialog(name,table[,parent])
```

Where **name** is the name by which you'll refer to the dialog, **table** is the dialog table name used to create dialog (see below), and **parent** is the parent window of the dialog, this can be a **window name**, or -1 = Desktop window, -2 = Main mIRC window, -3 = Currently active window, -4 = Currently active dialog, if no dialog is open, defaults to -3 behaviour.

Note: This type of dialog cannot be called from a remote script event.

You can also use the **$dialog(name/N)** identifier to list any open dialogs, where N returns the Nth open dialog. If N is zero, the total number of open dialogs is returned.

$dialog() also supports these **properties**:

x,y,w,h	returns the size and position of the dialog
cw,ch	returns width and height of client area of dialog
title	returns the title of the dialog
modal	returns $true if the dialog is modal, otherwise $false
table	returns the dialog table that the dialog is using
ok	returns the id of the Ok button if you specified one
cancel	returns the id of the Cancel button if you specified one
result	returns the id of the Result button if you specified one
focus	returns id of control that currently has focus
tab	returns id of tab that is currently displayed
active	returns $true if dialog is the active window, otherwise $false
hwnd	returns dialog window handle

The dialog table

You can use the **dialog** prefix to create a dialog table called **name** in a script using this format:

```
dialog [-l] name {
   title      "text"
   icon       filename, index
   size       x y w h
   option     type (dbu, pixels, notheme)

   text       "text", id, x y w h, style   (right, center, nowrap)
   edit       "text", id, x y w h, style   (right, center, multi, pass, read, return, hsbar, vsbar,
autohs, autovs, limit N)
   button     "text", id, x y w h, style   (default, ok, cancel, flat, multi)
   check      "text", id, x y w h, style   (left, push, 3state)
   radio      "text", id, x y w h, style   (left, push)
   box        "text", id, x y w h, style
   scroll     "text", id, x y w h, style (top left bottom right horizontal range N N)

   list       id, x y w h, style              (sort, extsel, size, vsbar, hsbar)
   combo      id, x y w h, style              (sort, edit, drop, size, vsbar, hsbar)

   icon       id, x y w h, filename, index, style      (noborder top left bottom right)

   link       "text", id, x y w h

   tab        "text", id, x y w h
   tab        "text", id

   menu       "text", menuid [, menuid]
```

```
  item      "text", id [, menuid]
  item      break, id [, menuid]
}
```

The **-l** switch makes a dialog table local, so that it can only be accessed by other scripts in the same file.

Where **"text"** is the default text for a control, **id** is a number that uniquely identifies a control, **x y w h** are the position and size of the control, and **style** consists of a combination of the words in brackets.

Other Styles
In addition to the styles shown in **brackets** for each type of control, you can also specify these styles:

disable	disables the control
hide	hides the control
group	start of a group
result	identifies the control whose value will be used as the return value to the calling script when the user presses the **ok** button

Tab control
The first **tab** definition specifies the **size** of the tab control, and any following **tab** definitions add tabs to it. You can **associate** controls with a specific tab by using the **tab** style followed by the **id** value for a tab, as shown in the example below.

```
dialog test {
  title "mIRC"
  size -1 -1 110 100
  option dbu

  tab "m", 1, 5 5 100 90
  tab "I", 2
  tab "R", 3
  tab "C", 4

  button   "m is for ... ;)", 11, 30 50 50 24, ok tab 1
  button   "I is for Internet", 12, 30 50 50 24, tab 2
  button   "R is for Relay", 13, 30 50 50 24, tab 3
  button   "C is for Chat", 14, 30 50 50 24, tab 4
}
```

You can use /did -fu to set the focus on a specific tab, and /did -vh to show/hide the entire tab control.

Menus
You can add menus and submenus to custom dialogs using the **menu** and **item** prefixes:

```
dialog name {
  menu "text", <menuid> [, menuid]
  item "text", <id> [, menuid]
  item break, <id> [, menuid]
}
```

If you **don't** specify a **menuid** for an item, it will use the **last** menuid that was created/used in a previous menu/item.

For example, the following menu definition reproduces the **File menu** of the mIRC Editor dialog:

```
dialog test {
   title "mIRC"
   size -1 -1 110 100
   option dbu

   menu "&File", 60
   item "&New", 70

   menu "&Load", 80, 60
   item "&Script", 90
   item break, 100
   item "&Users", 110
   item "&Variables", 120

   item "&Unload", 130, 60

   item break, 140

   item "&Save", 150
   item "&Save As...", 160
   item "Save &All", 170

   item break, 180

   item "Save && &exit", 190, ok
   item "&Cancel", 200, cancel
}
```

You can use the **/did** command (described below) to enable/disable, check/uncheck, append, delete, insert, and overwrite a menu item.

To **add** an item to a menu you can use: /did -a name <menuid> <newid> <text>.

To **insert** an item you must use: /did -i name <id> <newid> <text> where <id> is the item before which you want to insert an item.

Dbu vs. Pixels
The **dbu** option makes mIRC use dialog base units when creating the dialog, this ensures that the dialog will look the same for all users under any size display, etc. **Pixels** is the default however to maintain compatibility with previous scripts.

The identifiers **$dbuw** and **$dbuh** return the dbu per pixel width and height values for the current display. This may or may not be an integer value.

Ok and Cancel buttons
You must specify an **ok** and/or **cancel** button. When a user clicks the ok or cancel buttons, the dialog is closed. You can prevent the dialog from closing if a user clicks the ok button by using /halt.

Default position and size

If you specify -1 for any of the x y w h values in the size setting for the dialog, a default setting is used. To make mIRC center the dialog in a window, specify x y as -1 -1. The size setting can be over-ridden in the **init** event (see below) by using /dialog -s name x y w h.

The On Dialog event

If a user changes the state of controls in the dialog, eg. clicks on a button, types text in an edit box, etc., this triggers the on dialog script event which allows you to monitor input from the user:

```
 on 1:dialog:name:event:id: {
    echo $dname $devent $did
 }
```

Where **name** identifies the dialog, **id** is the id number of the control triggering the event, and **event** can be:

init	just before a dialog is displayed, controls can be initialized in this event. id is zero.
edit	text in editbox or combo box changed
sclick	single click in list/combo box, or check/uncheck of radio/check buttons, or click of a button
dclick	double click in list/combo box
menu	a menu item was selected
scroll	scroll control position has changed

You can also detect **mouse events** that aren't associated with a specific control:

mouse	mouse moved
sclick	left button down
uclick	left button up
dclick	double click
rclick	right button click
drop	drop click

You can use $mouse to retrieve the current mouse position.

The /did command

The /did command allows you to **modify** the values of **controls** in a dialog, eg. changing the text in an edit control, or setting focus to a button, or deleting lines in a listbox.

```
   /did -ftebvhnmcukradiogj name id [n] [text | filename]
```

-f	set focus on id
-t	set id as default button
-e	enable id
-b	disable id
-v	make id visible
-h	hide id
-n	enables editbox
-m	disables editbox
-c	check checkbox/radiobutton list/combo line
-u	uncheck checkbox/radiobutton list/combo line
-k	works with -cu, keeps other selections in a listbox

-r	clear all text in id
-a	add line of text to end
-d	delete Nth line
-i	insert text at Nth line
-o	overwrite Nth line with text
-g	set a new icon/bmp to an icon control /did -g name id [n] filename
-z	resets the width of horizontal scrollbar in listbox
-j	resets the edited setting in an editbox

Where **name** identifies the dialog and **id** is the id number of the control you want to modify.

If you want to modify **several** controls at the same time, you can specify **multiple** id numbers separated by commas, eg. /did -b name 2,12,14,16 etc.

You can select a range of text in an editbox using /did -c name id [n] [start [end]]. This selects line N in editbox, and sets selection to specified range of characters.

You can mark a **3state** checkbox as indeterminate by specifying both **-cu** switches.

You can access the **edit control** in a combobox by using 0 as the N value.

To change the **range** of a scrollbar control, you can use /did -z name id [min max].

$did(name,id)

You can get the settings and values of controls in a dialog by using the **$did()** dialog id identifier with the following formats and properties as appropriate to the control to which you are referring.

$did(name,id)
$did(name,id,N)

If used in the on dialog event, **name** is optional.

The following $did() **properties** are supported:

text	returns line or Nth line $did(id) is same as $did(id).text
len	returns length of line or length of Nth line
lines	returns number of lines
sel	returns Line Number of Nth selected line if N is 0, returns number of selected lines
seltext	returns selected text in an editbox or first selected item in a listbox
selstart	returns select start character in editbox line
selend	returns select end character in editbox line
edited	returns $true if text in editbox was changed
state	returns 0 = off, 1 = on, and 2 = indeterminate
next	returns id of next control in tab key order
prev	returns id of previous control in tab key order
visible	returns $true if the control is visible, otherwise $false
enabled	returns $true if the control is enabled, otherwise $false

You can access the edit control in a combobox by using 0 as the N value.

$didwm(name,id,wildtext,N)
Returns the number of the line that matches wildtext, with the search starting at line N. N is optional.

$didtok(name,id,C)
Returns a tokenized list of the items in a list/combo/edit box.

You can use **/didtok name id C text** to add a tokenized list of items to a list/combo/edit box.

Agents

mIRC supports Microsoft Agent either through scripting or through the Agents section in the options dialog. An **agent** is an animated character that can speak text and perform actions.

You can find links to related websites and resources, as well as download information, on the mIRC website at http://www.mirc.co.uk/agents.html

Agent commands

The following commands allow you to manipulate agents, to make them speak, play animations, and more.

/gload <-h> <name> <filename | N | default>
You must use /gload to load an agent before you can use it.

The **name** you give an agent is the name you will use to refer to it in all of the other commands and identifiers.

You can load an agent either by specifying its **filename**, if you know it, or by loading the **Nth** installed agent on your system, or by specifying **default**, which will load the default agent for your system.

If you specify the **-h** switch, mIRC will hide the agent whenever mIRC is minimized. You can also use the **-h** switch with the other agent commands to prevent them from popping the agent when mIRC is minimized and has the **-h** setting.

Note: You can't load more than one of the **same** agent. You can however load as many **different** agents as you want.

/gunload <name>
This unloads the specified agent. The **name** is the name you gave your agent when you loaded it with /gload, not the filename of the agent.

/gshow <name> [x y]
This shows an agent in its most recent or default position, or at the specified x y position.

/ghide <name>
This hides an agent.

/gmove <name> <x> <y> [speed]
This moves an agent to the <x> and <y> position on your screen. If a speed isn't specified, it uses a default speed. If you specify a speed of 0, it moves instantly to the new position.

/gsize <name> <w> <h>
This resizes an agent to the specified width and height.

/gtalk -kwlu <name> <text | <wavefile | text>>
This makes an agent speak the specified text.

If you want the agent to **think** the text in a balloon without speaking it, you can use the **-k** switch.

If you want the agent to play a **wave** sound file, you can use the **-w** switch, and specify a wave **filename**. You should also specify **text** after the filename, the agent will show it in a ballon while playing the wave sound.

The **-l** switch applies the lexicon settings in the lexicon dialog to the text.

The **-u** switch applies the speech settings in the speech options dialog.

/gplay <name> <anim | N> [timeout]
This makes an agent play one of its animations.

You can either specify the name **anim** of an animation, if you know it, or ask it to play the **Nth** animation.

Some animations are **looping** animations, which repeat continuously, and prevent an agent from doing anything else until you **/gstop** the agent. The **[timeout]** value allows you to specify a timeout for an animation after which it is **automatically** stopped, and the agent will then perform any **remaining** queued requests. If you don't specify a timeout value, the default is **5 seconds**. If no play or talk requests are pending, the looping animation continues beyond the timeout until there are.

/gpoint <name> <x y>
This makes an agent point towards the specified <x> <y> screen position.

/gstop -c <name> [talk play]
This stops an agent from doing what it's currently doing, and removes all queued requests for the agent.

If you wish to only stop the **current** action, you can use the **-c** switch.

If you wish to stop only **talk** or only **play** requests, you can specify talk or play.

/gopts -bieqnh <name> <on off size pace hide nosize nopace nohide langid>
This sets various options relating to how the agent behaves.

The **-b** switch turns balloons **on** or **off**. You can also make ballons **size** to fit the text being spoken, **pace** to display text word by word as it is being spoken, and **hide** to hide the balloons when no text is being spoken.

The **-i** switch turns idle effects **on** or **off**.

The **-e** switch turns sound effects **on** or **off**.

The **-n** switch allows you to set a language id, where **langid** is the hex id value.

The **-h** switch turns the auto-hide feature **on** or **off** (see /gload for more info).

/gqreq <on | off>
By default mIRC queues all requests and plays them one after the other to ensure that even if you use multiple agents at the same time, all messages will be heard, and all agents will act in the order that you requested. You can turn queuing on/off on the fly.

Agent Identifiers

The following identifiers allow you to access information about an agent that is currently loaded.

$agentver
Returns the version of the Agent installed on your system, 0 if not installed.

$agentstat
Returns 1 if Agent is ready, or 0 if busy/speaking.

$agentname
Returns the name of the agent in an on AGENT event.

$agent(N).char
Returns the filename of the Nth available agent installed on your system.

If you specify 0, returns total number of installed agents.

$agent(name)
Returns information about a currently loaded agent.

Properties: name, fname, visible, x, y, w, h, ow, oh, speed, pitch, idle, effects, active, langid, balloon, hide

name	the name you gave to this agent
fname	the filename of the agent or default
visible	returns $true or $false
x,y,w,h	left/top position, width/height.
ow, oh	original width/height
speed	speaking speed
pitch	speaking pitch
idle	if idle behaviour is on or off
effects	if sound effects are on or off
active	if this agent is active/topmost
langid	language id of system
balloon	current setting: on off size pace hide
hide	returns auto-hide setting

$agent(name,N)
Returns information on animations and queued lines for an agent.

Properties: anim, line

anim	returns the names of the animations available for this agent. If you specify N, returns the name of the Nth animation. If you specify 0, returns the total number of animations.
line	returns the list of lines currently queued for talking for this character. If you specify N, returns the Nth line. If you specify 0, returns total number queued lines.

Tags in spoken text
You can use tags in <text> in /gtalk which may be recognized by the text-to-speech engine, these are a few:

\spd=n\	set speed of spoken text
\pit=n\	set pitch of spoken text
\vol=n\	set volume of spoken text

\chr="text"\	where text is normal, monotone, or whisper
\ctx="text"\	where text is address, email, unknown
\emp\	emphasize next word
\pau=n\	pause speech nnn milliseconds
\rst\	reset settings to default

$notags(text)
Removes the above tags from text. Only tags that are valid and correctly written as above are removed.

Voice Commands

If you have **Speech Recognition** software installed, mIRC can be made to listen to **voice commands** through scripting.

/vcmd -lc <on | off | sleep>
This turns voice command listening **on** or **off**, or you can temporarily turn off listening by specifying **sleep**.

The **-c** switch clears the commands list.

The **-l** switch lists the commands in your commands list.

Note: Your SR software may have very large speech files or dictionaries which could make it slow to load/unload and/or process your speech.

/vcadd <command1,command2,...>
This adds voice commands to the commands list.

Commands should consist of at least **two words** and should be sufficiently **distinct** from other commands to make it easier for your SR software to match the words you speak to commands in your commands list.

Note: Adding or removing commands can be done on the fly, however if your SR software is slow at updating the commands list, it could cause short pauses in mIRC.

/vcrem <command1,command2,...>
This removes commands from the commands list.

$vcmdver
Returns the version of your installed SR software, or $null if not installed.

$vcmdstat
Returns 0 if not available, 1 if currently off, 2 if ignoring commands, 3 if listening for a command.

$vcmd(N)
Returns the Nth item in your commands list.

The on VCMD event
If the SR software matches a word you've spoken to a word in your commands list, it triggers the on VCMD event:

```
  on 1:VCMD:<matchtext>:<*/#/?/@>:/echo 3 Recognized: $1-
```

Example Script

```
alias vctest {
  if ($vcmdver == $null) halt
  vcmd -c on
  vcadd connect Dalnet, connect Efnet, connect Undernet, connect IRCnet
  vcadd Part Channel, Disconnect, List Commands, Moo Cow
}
```

```
on 1:vcmd:connect*:*:server $2
on 1:vcmd:part channel:*:if ($active ischan) part $active
on 1:vcmd:disconnect:*:quit
on 1:vcmd:list commands:*:vcmd -l
on 1:vcmd:moo cow:*:splay moo.wav
on 1:vcmd:*:*:echo You said: $1-
```

Hash Tables

Hash tables allow you to efficiently store large amounts of information which can be quickly referenced and retrieved later on.

A hash table can be created, freed, referenced, or modified using the following commands and identifiers.

/hmake -s <name> <N>
Creates a new hash table with N slots.

A hash table can store an **unlimited** number of items regardless of the N you choose, however the bigger N is, the faster it will work, depending on the number of items stored.

eg. if you expect that you'll be storing **1000** items in the table, a table of N set to **100** is quite sufficient.

The **-s** switch makes the command display the result.

/hfree -sw <name>
Frees an existing hash table.

The **-w** switch indicates that **name** is a wildcard, all matching tables are freed.

/hadd -smbczuN <name> <item> [data | &binvar]
Adds an item to an existing hash table.

If the item you're adding already exists, the old item is replaced.

The **-m** switch makes /hadd create the hash table if it doesn't already exist.

The **-uN** switch unsets the item after N seconds.

The **-b** indicates that you're adding a &binvar item to the hash table.

The **-c** switch chops the &binvar up to the first null value and treats it as plain text.

The **-z** switch decreases hash item once per second until it reaches zero and then unsets it.

The **/hinc** and **/hdec** commands use the same parameters as **/hadd** and increase or decrease the number value of an item.

When used with /hinc or /hdec, the **-c** switch increases or decreases the value once per second.

/hdel -sw <name> <item>
Deletes an item from a hash table.

The **-w** switch indicates that **item** is a wildcard, all matching items are freed.

/hload -sbn <name> <filename>
/hsave -sbnoau <name> <filename>
Load or save a table to/from a file.

These load/save **plain text** to a text file, with item and data on **separate** lines. $cr and $lf characters are **stripped** from text when saving as plain text.

The **-b** switch loads or saves binary files. $cr and $lf are preserved when saving as binary files.

You can use **-n** to load or save files as data only, with no items. When loading with **-n** each line of data is assigned an N item value, starting at N = 1.

/hsave also supports **-o** to overwite an existing file, and **-a** to append to an existing file.

By default /hsave excludes items that are in the /hadd -uN unset list, the **-u** switch forces it to include the unset items.

Note: /hload does not create the table, it must already have been created by /hmake.

$hget(name/N)
Returns name of a hash table if it exists, or returns the name of the Nth hash table.

Properties: size

$hget(moo).size	returns the N size of table, as specified in /hmake

$hget(name/N, item)
Returns the data associated with an item in the specified hash table.

Properties: unset

The **unset** property returns the time remaining before an item is unset.

$hget(name/N, item, &binvar)
Assigns the contents of an item to a &binvar.

$hget(name/N, N).item
This allows you to reference the table as an index from 0 to N, in order to look up the Nth item in the table.

If N is zero, returns the total number of items in the table.

You can also reference the Nth data value directly with $hget().data.

Note: This method is provided as a convenience, it is not an efficient way to use the hash table.

$hfind(name/N, text, N)
Searches table for the Nth item name which matches text. Returns item name.

Properties: data

If you specify the .data property, searches for a matching data value.

$hmatch(name/N, text, N)
Searches table for the Nth item name which matches text, where either text or item/data is wildcard text. Returns item name.

Properties: data

If you specify the .data property, searches for a matching data value.

$hregex(name/N, re, N)
Searches table for the Nth item which matches regular expression. Returns item name.

Properties: data

If you specify the .data property, searches for a matching data value.

COM Objects

mIRC allows you to call **COM objects** via scripts. You **must** have experience with COM objects in order to use this feature.

/comopen name progid
This opens a COM connection to object progid eg. Excel.Application, and assigns the connection a name.

You should check **$comerr** after making this call to confirm that the COM conection was successful.

/comclose name
This closes the specified COM connection.

/comreg -u filename
This registers/unregisters a COM DLL with windows.

```
example {
  comopen name progid

  ; if comopen failed, maybe the DLL that came with the script isn't registered
  if ($comerr) {

    ;register the DLL
    comreg test.dll

    ;try to open it again
    comopen name progid

    ; still failed, halt the script
    if ($comerr) halt
  }
}
```

$comerr
This should be checked after a call to any COM command or identifier. Returns 1 if there was an error, 0 otherwise.

$com(name,member,method,type1,value1,...,typeN,valueN)
This calls a member of an open COM connection with the specified method and parameters.

name - connection name.

member - member name.

method - Combination of the following values added together:
1 = DISPATCH_METHOD
2 = DISPATCH_PROPERTYGET
4 = DISPATCH_PROPERTYPUT
8 = DISPATCH_PROPERTYPUTREF

type - the variable type, can be: i1, i2, i4, ui1, ui2, ui4, int, uint, r4, r8, cy, date, decimal, bool, bstr, variant, dispatch, unknown, error.

VB equivalents are: boolean, byte, currency, date, double, integer, long, single, string, variant.

To make a variable by reference, use * in the type name, eg. i1*

To assign a name to a variable for later reference after a call, append it to the type, eg. i1* varname

When using a variant you must also specify the variable type after it, eg. variant bool.

value - the value assigned to the variable type.

Returns: 1 = ok, 0 = fail.

After you've opened a COM conection or made a call to $com() you can use the following forms of $com():

$com(name/N)
Returns name if connection is open, or name of Nth connection. N = 0 returns number of open connections.

Properties: progid, dispatch, unknown, result, error, errortext, argerr

progid - object name.

result - the value returned by the COM object member after the call.

error - error value, if there was an error.

errortext - error description associated with error.

argerr - Nth argument that caused the error, if the error was due to an invalid variable type.

$com(name/N,varname)
Returns value of the specified variable name.

Dispatch and Unknown

The two variable types **dispatch** and **unknown** allow you to **pass** dispatch/unknown pointers as parameters in a $com() call, or **retrieve** dispatch/unknown pointers from a $com() call, by reference.

To **pass** a dispatch/unknown pointer as a parameter in $com(), specify the variable type as dispatch/unknown, and specify the name of an existing $com() connection as the value.

To **retrieve** a dispatch/unknown pointer through a call to $com(), specify the variable type as dispatch/unknown with *, and assign it a variable name. When $com() returns, mIRC will create a new $com() item with that variable name and assign it the dispatch or unknown pointer.

In the case of retrieving an **unknown** pointer, mIRC will extend it to a **dispatch** pointer if it can, allowing you to call it directly via $com().

You can use $com().dispatch or $com().unknown to see if a pointer exists for that $com() item.

Example Script

The following script is a simple example that connects to excel and then retrieves and sets the visible property.

```
excel {
  comopen excel Excel.Application

  if ($comerr) {
    echo comopen failed
    halt
  }

  ; check if excel window is visible

  if ($com(excel,Visible,3) == 0) {
    echo $com failed
    goto finish
  }

  echo Visible: $com(excel).result

  ; make excel window visible

  if ($com(excel,Visible,5,i4,1) == 0) {
    echo $com failed
    goto finish
  }

  ; check visibility again

  if ($com(excel,Visible,3) == 0) {
    echo $com failed
    goto finish
  }

  echo Visible: $com(excel).result

  :finish
  comclose excel
}
```

If-then-else statements

The **If-then-else** statement allows you to **compare** values and **execute** different parts of a script based on that comparison.

Basic format

if (v1 operator v2) { commands }
elseif (v1 operator v2) { commands }
else { commands }

The **( ) brackets** enclose **comparisons**, whereas the **{ } brackets** enclose the **commands** you want to be performed if a comparison is true. You must make sure that the number of **( ) and { }** brackets match to make sure that the correct comparisons are made, and that the correct commands are executed.

Using brackets **speeds up** processing. If an alias uses **too few** brackets then the statement might be **ambiguous** and the alias will take longer to parse, might be parsed incorrectly, or might not be parsed at all.

You can **nest** as many if-then-else statements as you want inside each other.

The Operators

==	equal to
===	equal to (case-sensitive)
!=	not equal to
<	less than
>	larger than
>=	larger than or equal to
<=	smaller than or equal to
//	v2 is a multiple of v1
\\	v2 is not a multiple of v1
&	bitwise comparison
isin	string v1 is in string v2
isincs	string v1 is in string v2 (case sensitive)
iswm	wildcard string v1 matches string v2
isnum	number v1 is a number in the range v2 which is in the form n1-n2 (v2 optional)
isletter	letter v1 is a letter in the list of letters in v2 (v2 optional)
isalnum	text contains only letters and numbers
isalpha	text contains only letters
islower	text contains only lower case letters
isupper	text contains only upper case letters
ison	nickname v1 is on channel v2
isop	nickname v1 is an op on channel v2
ishop	nickname v1 is a halfop on channel v2
isvoice	nickname v1 has a voice on channel v2
isreg	nickname v1 is a normal nick on channel v2
ischan	if v1 is a channel which you are on.
isban	if v1 is a banned address in internal ban list

isaop	if v1 is a user in your auto-op list for channel v2 (v2 optional)
isavoice	if v1 is a user in your auto-voice list for channel v2 (v2 optional)
isignore	if v1 is a user in your ignore list with the ignore switch v2 (v2 optional)
isprotect	if v1 is a user in your protect list for channel v2 (v2 optional)
isnotify	if v1 is a user in your notify list.

To **negate** an operator you can prefix it with an **!** exclamation mark.

$ifmatch
Returns the first parameter of matching if-then-else comparison. So, in the case of this comparison:

```
if (text isin sometext) { ... }
```

$ifmatch will return "text"

Combining comparisons

You can combine comparisons by using the **&& for AND** and **|| for OR** characters.

```
number {
  if (($1 > 0) && ($1 < 10)) {
    if ($1 < 5) echo Number is less than five
    else echo Number is greater than five
  }
  else echo Number is out of bounds
}
```

This alias checks if the number you specify, when you type /number <value>, lies within the required range.

The ! not prefix

You can use the **!** prefix in front of variables and identifiers in an if-then-else statement to negate a value. The following statements are equivalent.

```
if (%x == $null) echo x has no value

if (!%x) echo x has no value
```

Examples

```
listops {
  echo 4 * Listing Ops on #
  set %i 1
  :next
  set %nick $nick(#,%i)
  if %nick == $null goto done
  if %nick isop # echo 3 %nick is an Op!
  inc %i
  goto next
  :done
  echo 4 * End of Ops list
}
```

This alias lists the Ops on the current channel. It does this the hard way since we could just use $opnick() instead but using $nick() serves as an example of how **isop** can be used and how **$null** is returned once we reach the end of the list.

```
GiveOps {
  %i = 0
  %nicks = ""
  :nextnick
  inc %i
  if ($snick(#,%i) == $null) { if ($len(%nicks) > 0) mode # +oooo %nicks | halt }
  %nicks = %nicks $snick(#,%i)
  if (4 // %i) { mode # +oooo %nicks | %nicks = "" }
  goto nextnick
}
```

This is a popup definition which Ops the nicknames which are selected in the current channel nicknames listbox.

```
on 1:ctcpreply:PING* {
  if ($2 == $null) halt
  else {
    %pt = $ctime - $2
    if (%pt < 0) set %pt 0
    if (%pt < 5) echo 4 [PING reply] $nick is too close for comfort
    elseif (%pt < 20) echo 4 [PING reply] $nick is at just about the right distance
    else echo 4 [PING reply] Earth to $nick earth to $nick
  }
  halt
}
```

This intercepts a ping reply and prints out a silly message based on how far away the person is.

Multi-server

mIRC allows you to connect to more than one IRC server at a time. This means that scripts need to be multi-server aware in order to behave correctly when a user is connected to more than one server. The following **commands** and **identifiers** allow a script to handle multiple server connections.

Identifiers

Each **new server window** that is created is assigned a **connection id**.

Each **window** that is created, such as a channel or query window, is associated with the **connection id** of the server where that window was opened.

$cid
Returns the server **connection id** for the current script.

Some window identifiers have their own **connection id** counterpart, such as **$activecid** and **$lactivecid**.

Most window related identifers also have a **.cid** property.

$scid(N)[.id]
Returns the connection id, where N is a $cid value.

If N = 0, returns total number of open server windows.

If you specify a **property** which is an identifier, it returns the value of that identifier for that connection. This also works for **custom** identifiers.

Note: The property cannot use brackets.

$scon(N)[.id]
Returns the connection id, where N is the Nth connection.

If N = 0, returns total number of open server windows.

If you specify a **property** which is an identifier, it returns the value of that identifier for that connection. This also works for **custom** identifiers.

Note: The property cannot use brackets.

Commands

Scripts can be made to perform commands on specific server connections by using **/scid** and **/scon**.

/scid <-rsatM | N> [command]
Changes the active connection for a script to connection id N, where N is a $cid value.

All commands **after** the /scid command will be performed on the new connection id.

The **-r** switch resets the connection id to the original id for that script.

If you specify the **command** parameter, the connection id is set only for **that** command.

The **/scon** command works in exactly the same way, except that N represents the **Nth** connection, not a connection id value.

The **-a** and **-tM** switches can only be used if you specify a command.

The **-a** switch performs the command on all connection ids.

The **-tM** switch limits the command to being performed only on servers with a certain connection status, where M is an or'd value of 1 = server connected, 2 = not connected, 4 = connecting, 8 = not connecting.The command is only performed if M matches the connect status of the connection id.

Note: If you use a command that contains $identifiers, and you want the identifiers to be evaluated in the target connection, you must pass them as $!identifier to prevent them from being evaulated first in the current connection.

The Internal Address List

mIRC maintains an **internal address list** of all users who are currently on the same **channels** as you.

This address list is used by the /guser, /ruser, /ban, /ignore, /finger, and /dns commands to quickly find a user's address without resorting to a **/userhost** server lookup.

A user's address is **added** to the list either when they join the channel, send a message to a channel, or make a mode change.

A user's address is **removed** from the list when they are no longer on any of the channels which you are currently on.

The reason why only addresses for users on the **same channels** as you are stored is because this guarantees the **integrity** of the list. ie. that a specific nickname is associated with a specific address.

IAL Commands

/ial [on | off]
Turns IAL on or off. Note that this setting is not persistent across sessions and resets to **on** every time mIRC is run.

/ialclear [nick]
Clears the IAL, or if a nickname is specified, clears that nickname's IAL entry.

/ialmark <nick> [text]
Marks the IAL entry for a nickname with the specified text.

IAL Identifiers

$ial(nick/mask,N)
Returns the Nth address matching nick or mask in the Internal Address List.

Properties: nick, user, host, addr, mark

$ial(*!*@*.com,0)	returns the total number of addresses in the IAL matching *!*@*.com
$ial(*!*@*.com,3)	returns the 3rd address in the IAL matching *!*@*.com
$ial(*!*@*.com,4).nick	returns the nick of the 4th matching address ending in .com
$ial(*!*@*.com,4).user	returns the userid of the 4th matching address ending in .com

To scan each address in the IAL you can use $ial(*,N).

The N parameter is optional, defaults to 1 if not specified.

$ialchan(nick/mask,#,N)
Returns the Nth address on the specified channel matching nick or mask in the Internal Address List.

This works the same way as the $ial() identifier.

Connection Problems

The most **common** problems related to connecting to an IRC Server are described below.

Unable to resolve IRC Server
If you try to connect to a server and see this message, the problem could be:

Your Internet Provider's DNS isn't working
This happens occasionally and is a temporary problem with your Internet Provider. This problem would also result in your being unable to connect to other internet services such as web sites. You should try again later.

An Invalid or Non-working IRC Server address
You might be trying to connect to an IRC Server that is currently not working, or perhaps is an old address and doesn't exist anymore. You should try another IRC Server.

Unable to connect to IRC Server
If you try to connect to a server and see this message, the problem could be:

IRC Server isn't working
You might be trying to connect to an IRC Server that is currently not working. This is the most likely problem. You should try another IRC Server.

Invalid Port number
The IRC Server address might be correct but you've specified the wrong port. Most servers operate at least on port 6667, you should try that port to see if it solves the problem.

Other messages
If you try to connect to a server and get **Disconnected** and see the message **Closing Link** followed by a comment such as **No Authorization**, or **No More Connections**, etc., it might be that you're too far away geographically from that server, or that the server is full and can't handle anymore users, or there may be other reasons. You should try a different, closer IRC Server until one works for you.

Note: If you try to connect to a server and you get **Unable to resolve local host** then see the Local Info section.

Finger

This allows you to finger a persons address to find out more information about them.

If a user has a personal email/machine address, this will only work if they are running a finger server. Most other addresses eg. school, government, etc. can usually be fingered sucessfully, though they will not necessarily give you any useful information. For some users this can be a way of checking if they have mail.

You can also type **/finger <nick/address>** at the command line.

If there's a **.** in the parameter you provide then it assumed to be an address and is immediately fingered, otherwise it is assumed to be a nickname, so mIRC looks up the users address with a /userhost and then fingers that address.

On a related note, check out the Finger Server.

Online Timer

The **Online timer** is displayed in the titlebar of the status window next to your connection information.

You can choose to have the timer **reset to zero** every time you connect, so that you can tell how long you are on for that session, or if you select the second option, the timer will be **cumulative** and will show the **total** amount of time you have used IRC since the last reset.

Address Book

The Address Book can be used to store various types of information about users. It can be accessed via the Tools menu, the toolbar, or by pressing **Alt+B**, or by using the **/abook [nickname]** command.

Address

The **Address** section allows you to store basic information about users on IRC.

Nickname
Nickname on IRC. Information in the Address Book is stored according to nickname.

Real Name
Real name.

Email
Email address. If you click on the **Email** button, mIRC will start up your email software.

Website
Website address. If you click on the **View** button, mIRC will start up your web browser.

IP Address
IP Address. If you click the Chat button, mIRC will initiate a DCC Chat directly to this user's DCC Server instead of using the IRC Server.

Note: A user may not always have the same IP Address.

Notes
Notes. You can enter various notes about a user.

Picture
If a user sends you their picture, you can associate it with their nickname in the address book.

Info

The **Info** section displays the result of the /uwho command which looks up information for a user on IRC. The format of the /uwho command is:

/uwho [nick] [nick]

This performs a **/whois** on the specified nickname to look up their server information and then displays it in the info section of the address book. Because of the way IRC Servers work, you may need to specify the nickname a second time in order to look up information such as idle time or the away message, however this information usually takes longer to retrieve.

Note: The address shown in the **Info** dialog may not be an email address, it is mainly an indication of the user's internet provider.

Notify

The Notify list is a like a buddy list, it notifies you whenever a nickname is on IRC.

Control

The Control section performs functions related to channel and user control.

Nick

The Nick colors section allows you to assign colors to nicknames.

Notify List

The **notify list** is a like a buddy list, it notifies you whenever a nickname in your notify list **comes on** or **leaves** the IRC network you are on.

Note that the notify list can work differently depending on the IRC network you are using. On some networks, the notify list is only updated very 40 seconds or so.

Adding a User

To add a user you can enter the following information and then click on the **Add** button.

Nickname
The nickname of a user that you want notify to look for.

Note
An optional note or reminder that will appear next to the users nickname.

Play sound
The sound that you want played whenever the user joins IRC.

Perform /whois
This option makes mIRC perform a **/whois** on the user when they join IRC to look up their address. You should only use this option if you really need to, if you use it with too many nicknames then the IRC server might disconnect you for flooding.

Add/Update/Delete
To add or delete users from the list. After you make a change to the settings of an existing user, you must click the **Update** button.

Notify display options

Show notifies in active window
The default is to show notifies in the status window, however checking this option will also show notifications in the current active window.

Only show notifies in notify window
This will make mIRC display notifies only in the notify window.

Pop up notify window on connect
This will pop up the notify list window when you connect to an irc server.

Show address & time
On IRC networks that support this feature, mIRC will display the nicknames address and time online.

The /notify command

You can also add/remove nicknames from the notify list by using the /notify command.

/notify [-shrl] <on|off|nickname> [note]

You can turn notify on and off by typing **/notify on or off** respectively.

The **-sh** switches can be used to **show or hide** the notify list window respectively

The **-r** switch removes the specified nickname from your notify list.

The **-l** switch displays your notify list.

The **note** is optional and allows you to specify a little note for each nickname.

If you prefix a nickname with a **+** sign then mIRC will do a **/whois** on the nickname as part of the notify. However, if you do this on too many nicknames then the IRC server might disconnect you for flooding, so it's best to use it only if you really need to.

You can manually force mIRC to update the notify list by typing **/notify** with no parameters.

Note: Some IRC networks might let you use a full address instead of just a nickname, the only way to see if it works is to try it out.

Control

The **Control** dialog performs functions related to channel and user control.

Auto-Op

If a user joins a channel where you have Ops and that user's address is listed in the auto-op list, they will be given Op status. You can add an address to the list in the following format:

nick!userid@host,#channel1,#channel2

On IRC, user addresses are specified in the format:

nick!userid@host

So if my nickname is **MadGoat** and my address is **khaled@mirc.com** then to put me in your list, you would use:

madgoat!khaled@mirc.com

If I change nicknames a lot, then you would use:

***!khaled@mirc.com**

If I change my nickname and userid a lot, then:

***!*@mirc.com**

The /aop command

/aop [-rw] <on|off|nick/address> [#channel1,#channel2,...] [type]

The **-r** switch indicates that the address is to be removed.
The **-w** switch makes the auto-op apply to any network.

If you do **not** specify a type then only the users nickname is used. If you specify a type then the users address is looked up via the server.

The **$aop** identifier returns $true if auto-op is enabled, and $false if it isn't.

The **$aop(address/N)** identifier returns any matching address in the list, or the Nth address. The **.type** property returns the list of channels.

Auto-Voice

The **auto-voice** list works in exactly the same way as the auto-op list. The **/avoice** command, which uses the same format as **/aop**, can be used to add or remove users to your auto-voice list.

The **$avoice** identifier returns $true if auto-voice is enabled, and $false if it isn't.

The **$avoice(address/N)** identifier returns any matching address in the list, or the Nth address. The **.type** property returns the list of channels.

Random delay auto-op/voice

This option introduces a random 1 to 7 seconds delay in the auto-op/voice routine. This is to prevent channel windows from filling up with mode notifications whenever a nickname is in the auto-op/voice list of several users. If at the end of the random delay the user has already been opped/voiced then mIRC does not perform an op/voice.

Ignore

This feature allows you to ignore various types of messages from users. You can either choose to ignore a user completely, or to ignore only specific messages from a user.

nick!userid@host,private,invite,ctcp

The /ignore command

/ignore [-lrpcntikdwxu#] <on|off|nick/address> [type]

Where p = private, c = channel, n = notice, t = ctcp, i = invite, k = control codes, d = dccs.

The **-u#** switch specifies a delay in seconds after which the ignore is automatically removed.
The **-r** switch indicates that the address is to be removed.
The **-x** switch indicates that this address should be excluded from ignores.
The **-l** switch displays the list of ignored addresses which match the specified switches.
The **-w** switch makes the ignore apply to any network.

If you do **not** specify a type then only the users nickname is used. If you specify a type then the users address is looked up via the server.

You can clear the ignore list by specifying **-r** with no address.

Note: If you have a /query window open with someone, private messages from them won't be ignored even if their address matches an ignore address.

The **$ignore** identifier returns $true if ignore is enabled, and $false if it isn't.

The **$ignore(address/N)** identifier returns any matching address in the list, or the Nth address. The **.type** property returns the ignore method.

Protect

If you are on a channel and you have channel Op status, any users that match the nicknames in the protect list will be automatically protected. mIRC does this by kicking or de-opping anyone who tries to kick or de-op your protected users. You can add an address to the list using the following format:

nickname,#channel1,#channel2

Note: This option is usually limited to using **nicknames** because of the way IRC servers work, however you can specify an address, and if the user is in your Internal Address List, they will be protected by address.

The /protect command

/protect [-rw] <on|off|nick> [#channel1,#channel2,...] [type]

The **-r** switch indicates that the address is to be removed.
The **-w** switch makes the protect apply to any network.

If you do **not** specify a type then only the users nickname is used. If you specify a type then the users address is looked up via the server.

The **$protect** identifier returns $true if protect is enabled, and $false if it isn't.

The **$protect(address/N)** identifier returns any matching address in the list, or the Nth address. The **.type** property returns the list of channels.

Nick Colors

The **Nick colors** section allows you to assign **colors** to **nicknames**, which are then highlighted with their assigned colors in the channel nicknames list, and in any messages that those nicks send to channel or query windows.

Adding a Nick

The **Add Nick** dialog is far simpler than it appears. You must select a **Nick color** from the color listbox, and **one** other item in the **Apply** section.

For example, you could select the color **red** and enter a nickname in the **Nick or Address** editbox. This would highlight that specific nickname in the color red in the channel nickname listbox, and in messages that this user sends to channel or query windows.

The settings in the **Apply** section are **cumulative**. This means that if you enter a nickname, and enter a channel mode, only users matching this nickname **and** having this channel mode will be highlighted.

The nick color list uses the first match it finds for any event, so you must **prioritize** the order of the items in the list yourself.

Note: You can specify %vars or $identifiers as the nick.

Commands and Identifiers

The nick color list can be modified and referenced by using the /cnick command and the $cnick() identifier.

/cnick -raniovpymN [nick[!user@host]] [color] [modes] [levels]
This allows you to modify the items in the nick color list.

The **r** switch removes the specified nick or address from the list.

You can use **/cnick -r nick/N** to remove first item that matches nick in the nick color list or the Nth item in the nick color list.

To add or refer to an item as **Any nick** you can use the ***** character as the nick.

The **a** switch sets the Any Mode option.
The **n** switch sets the No Mode option.
The **iovpy** switches set the ignore, op, voice, protect, and notify list options respectively.

The **mN** switch sets the highlight method, 0, 1, or 2.

The **color** item is the color you want to assign to the nick.
The **modes** item is the list of modes required for that item to match, eg. @%+
The **levels** item makes mIRC search your User List for a matching level and address.

Note: /cline over-rides the nick color list. You can use **/cline -r** to reset a nick to default color to make the nick color list apply to a nick.

$cnick(N/nick, M)
Returns Nth nick in nick color list, or if nick is specified returns Nth position of item in list that matches nick. If nick doesn't match any items, returns zero.

Properties: color, modes, levels, method, anymode, nomode, ignore, op, voice, protect, notify

To get a nick's color, you can use **$cnick(nick).color**. If nick doesn't match any items in the list, returns 'Normal Text' color, or if M = 1, returns 'Listbox text" color. M is optional.

List Channels

The List Channels dialog allows you to retrieve the list of currently active channels. You can view the List Channels dialog by clicking on the List Channels button in the toolbar. This dialog can be viewed either by clicking it's button in the toolbar or by typing key combination Alt+L.

Get List
This retrieves a list of all of the active channels from the IRC server. This list can be quite long and depending on your connection it might take several minutes to download. The IRC Server actually sends the whole list, regardless of the filters you specify. You will not be able to do anything on IRC until this retrieval has been completed.

mIRC allows you to specify different filenames for the channels list which can be useful if you regularly connect to different IRC networks.

Apply
This allows you to respecify the list parameters without having to retrieve the whole list again from the IRC server. Just change the parameters and then click on apply to have them relisted according to your new criteria.

Match text
You can enter several words (separated by spaces) which mIRC will look for in channel names. Only those channels which match any of the words you specify will be listed. If you leave this **empty** then all channels will be listed.

Match text in topics
If this is turned on then mIRC will apply the Match Text procedure to channel topics as well. So only channel topics that match any of the words in the Match text editbox will be listed.

Number of people on a channel
This allows you to limit the channels list to those channels which contain a number of people ranging between the specified minimum and maximum.

Lock/Unlock
This allows you lock the Hide parameters with a password thus preventing anyone from changing the Hide settings. The same password must be used to unlock this.

Hide channels which match...
You can enter several words (separated by spaces) which mIRC will look for in both channel names and topics. Any channels which match any of these words will be excluded from the channels list.

Hide non-alphanumeric channels
This will filter out any channels that begin with characters that aren't numbers or letters.

Hint: You can click your right mouse button in the Channels List window to pop up a menu with useful options.

Channel Central

The **channel central** dialog allows you to set various **channel modes** if you're an **Op** on a channel. If you're not an **Op** you will **not** be able to change the channel modes and they will appear **disabled**.

You can find out more about channel modes in the IRC Commands section.

Topic History
Lists the current topic, and the history of topic changes during your stay on a channel.

Bans List
This allows you to view, edit, or remove various types of mode settings, such as bans, exceptions, and invites. Most IRC networks support **ban** lists, however few support **exception** and **invite** lists currently.

Only Ops set topic
This allows only Ops on a channel to change the channel topic.

No external messages
This prevents users who are not on the channel from sending messages to the channel.

Invite only
This setting prevents users from joining the channel unless someone on the channel has specifically **invited** them using the /invite command, eg. /invite nickname #channel

Moderated
This prevents users from **speaking** on a channel unless they have been given a **voice** on the channel by an Op. This feature allows you to **moderate** the channel for events such as interviews, etc.

Key
This option allows you to set a **password** for the channel if you are an Op. Any user who wishes to join your channel will need to specify the password when joining, eg. /join #channel password

Limit to N users
This limits the number of people who can be on the channel at any one time to N people.

Private
This prevents a channel from being seen in a /whois unless the person issuing the /whois request is on the same channel. A channel that is +p will still show up in the channels list though.

Secret
This prevents a channel from being listed in the **channels list** retrieved by the List Channels dialog.

Note: The Events dialog for a channel can now be accessed via the System menu.

Playing Files

The **play** feature is a powerful tool that allows you to play **text** files to users or channels on IRC.

The **play central** dialog lists all of the currently **queued** play requests, and allows you to maintain the queue. Files are played in the order in which they are queued. The **play central** dialog can be displayed with the **/playctrl** command.

The /play command

A play request can be **added** to the queue by using either the **play** dialog or the **/play** command. The **play dialog** can be displayed by using the **/play** command with no parameters. It supports **most** of the features of the /play command itself, described below.

/play [-escpbn q# m# f# rl# t#] [channel/nick/stop] <filename> [delay]

In its **simplest** form, you can play a text file to the current window with:

/play c:\text\poem.txt

This plays the file poem.txt to the current window, which must be a query or channel window, with a default delay of 1000 milliseconds, ie. 1 second. Empty lines are treated as a delay.

If you have flood protection turned on, /play sends all lines through the flood protection feature to prevent you from **flooding** yourself off the server.

The **-e** switch allows you to echo out the text to a window as it would be sent to the server.

The **-s** switch allows you to play commands to the status window while offline. If you do not specify the -s switch then you must be connected to a server to use the /play command.

The **-c** switch forces mIRC to interpret lines as actual commands instead of plain text.

The **-n** switch makes the play command use /notice instead of /msg.

The **-p** switch indicates that this is a priority play request and should be placed at the head of the queue for immediate playing. The current play request will be paused and will resume once this play request is finished.

The **-q#** switch specifies the maximum number of requests that can be queued. If the queue length is already larger than or equal to the specified number then the play request is ignored.

/play -q5 c:\text\info.txt 1000

The **-m#** switch limits the number of requests that can be queued by a specific user/channel. If the user/channel already has or exceeds the specified number of requests queued then the play request is ignored.

/play -m1 info.txt 1000

The above line limits each user to a maximum of one request at a time and ignores all of their other requests.

Note: The **-q#** and **-m#** switches only apply to a /play initiated via a remote definition, not by you.

The **-b** switch plays text in the clipboard to a window. The text is temporarily saved to a file playqN.txt, which is deleted once playing is completed.

To combine the above switches you would use:

```
/play -cpq5m1 info.txt 1000
```

The **-r** switch forces a single line to be chosen randomly from a file and played.

```
/play -r action.txt 1500
```

The **-l#** switch forces the specified line-number to be read from a file and played.

```
/play -l25 witty.txt 1500
```

The **-f#** switch playes the whole file starting from the specified line.

```
/play -f9 moo.txt
```

For switches **-rlf** the first line in the file can be a single number specifying the number of lines in the file, this speeds up the process of reading the file.

The **-t** switch forces mIRC to look up the specified **topic** in the file and play all lines under that topic. For example:

```
/play -thelp1 help.txt
```

In the help.txt file you would have:

```
[help1]
line1
line2
line3

[help2]
...
```

mIRC will play everything after [help1] and stop when it reaches the next topic header or the end of the file.

You can use the **$pnick** identifier in commands which identifies the nick/channel to which you are currently playing.

To **stop** the playing of a file and clear the play queue you can use /play stop.

The $play identifer

The $play(N) or $play(Nick,N) identifer returns information on items in the play queue.

Properties: type, fname, topic, pos, lines, delay, status

If you specify a **nick**, you can find out how many play requests a user has in the queue.

Long File Names With.Spaces In Them

The DCC protocol doesn't take into account the possibility of a filename containing spaces, so most if not all IRC clients will incorrectly interpret the following DCC Send message:

PRIVMSG nick :DCC SEND This is a long file name with.spaces in it ipaddress port filesize

Thus mIRC gives you the **Fill Spaces** option; this fills spaces in a filename with the underscore character **_**, which should then allow other clients to interpret the message correctly. So other clients would see:

PRIVMSG nick :DCC SEND This_is_a_long_file_name_with.spaces_in_it ipaddress port filesize

If the Fill Spaces option isn't selected then mIRC sends **"Long File Names with.Spaces in them"** enclosed in quotes. For example:

PRIVMSG nick :DCC SEND "This is a long file name with.spaces in it" ipaddress port filesize

As far as I know, only mIRC can send and receive messages of this form (and only versions 3.8 and onwards), so if you try using this dcc send message with other clients it probably won't work.

Accepting Files on IRC and the Internet in General

Sharing files on IRC is part of what makes IRC fun, however it's important to be **careful** about **who** you accept files from and **which** types of files you accept.

Although **most** files are safe, there are always a few out there that may be infected with a virus, or may be malicious programs that try to damage your computer. Since it's impossible to know in advance whether a file that is being sent to you might cause a problem, following a few common sense rules usually helps to avoid problems:

1. Only accept files from people that you know and trust. You should **never** accept files from people you don't know, and **never** accept files without knowing what their purpose is.

2. Files ending in .BAT, .COM, .EXE, .DLL have the most potential to cause problems. You should **not** accept such files from people you don't know, or download them from web/ftp sites which don't appear trustworthy.

3. Aliases, Popups, or Scripts that can be **loaded** or **typed** into your IRC client can also cause problems. mIRC, and most other IRC clients, allow you to create scripts that perform **useful** functions, but these can also cause problems if misused. You should make sure that you know and trust the source of these files before using them.

4. Certain types of Document files can contain **macros** which are run by your Word Processor when you open the document to view it, so these are also potentially harmful. You should make sure that you have macro-warnings turned on in your Word Processer. It is also safer to view any documents that you receive in a plain-text editor first if possible.

5. If you have an anti-virus program, you should use it to scan all files that you download **before** you use them. However, IRC is a highly interactive medium where information spreads very quickly, so using an anti-virus program does not guarantee that a file will be safe since it takes time for anti-virus programs to be updated.

DCC Resume Protocol

The current protocol is very simple.

User1 is sending the file.
User2 is receiving the file.

To initiate a DCC Send, User1 sends:

PRIVMSG User2 :DCC SEND filename ipaddress port filesize

Normally, if User2 accepts the DCC Send request, User2 connects to the address and port number given by User1 and the file transfer begins.

If User2 chooses to resume a file transfer of an existing file, the following negotiation takes place:

User2 sends:

PRIVMSG User1 :DCC RESUME filename port position

filename = the filename sent by User1.
port = the port number sent by User1.
position = the current size of the file that User2 has.

User1 then responds:

PRIVMSG User2 :DCC ACCEPT filename port position

This is simply replying with the same information that User2 sent as acknowledgement.

At this point User2 connects to User1 address and port and the transfer begins from the specified position.

Note: The newer versions of mIRC actually ignore the filename as it is redundant since the port uniquely identifies the connection. However, to remain compatible with older mIRC's, mIRC still sends a filename as **file.ext** in both RESUME and ACCEPT.

DCC Server Protocol

Chat Protocol
Client connects to Server and sends:
100 clientnickname
When Server receives this, it sends:
101 servernickname
Connection is established, users can now chat.

Fserve Protocol
Client connects to Server and sends:
110 clientnickname
When Server receives this, it sends:
111 servernickname
Connection is established, user can now access fserve.

Send Protocol
Client connects to Server and sends:
120 clientnickname filesize filename
When Server receives this, it sends:
121 servernickname resumeposition
Where resumeposition is between 0 and filesize, and is required.
Connection is established, and Server dcc gets the file.

Get Protocol
Client connects to Server and sends:
130 clientnickname filename
When Server receives this, it sends:
131 servernickname filesize
When Client receives this, it sends:
132 clientnickname resumeposition
Where resumeposition is between 0 and filesize, and is required.
Connection is established, and Server dcc sends the file.

Other
If server receives unexpected information, or doesn't recieve info 15 seconds after initial connection, it closes the connection.

If service is unavailable, server sends:
150 unavailable

If server rejects connection, it sends:
151 rejected

Notes:
The Get protocol has been implemented in this way mainly because I'm assuming:
   1) The client may not be able to open a socket to listen for and accept a connection (firewall etc.)
   2) The DCC Server may only be able to listen for connections on port 59 (firewall etc.)
   3) Since the client was able to connect to the DCC Server the first time, it should have no problem connecting to the same port again.

Currently the Get Protocol is **only** used by the Fileserver when a user who has connected to a Fileserver via the DCC Server requests a "get filename". All other attempts to Get a file via the DCC Server are ignored.

DCC Socks5 Protocol

mIRC uses a **passive** protocol to establish DCC connections when a client is behind a SOCKS5 firewall.

The method will work with SOCKS5 firewalls that both:

i) Support listening/binding to a port of Zero for an incoming connection.
ii) Assign outgoing connections an IP address that's the same as the SOCKS firewall IP address.

The DCC Send/Chat CTCP messages are extended by adding an extra number to the end of the message which uniquely identifies the negotiation, and a port of Zero is specified to indicate that this is a passive connection request. All other fields are identical to a standard DCC Send/Chat message.

DCC Chat Passive Protocol

Client A, initiating the DCC Chat, sends the passive connection request below to Client B. The port is set to zero, and the id number is a unique integer identifying the connection:

```
   DCC CHAT nickname address port id
```

If Client A is behind a firewall, the address is the ip address of its SOCKS firewall.

Client B receives the message and sets up a listening socket, and sends the IP address and port back to Client A, specifying the id number that identifies Client A's request:

```
   DCC CHAT nickname address port id
```

If Client B is behind a SOCKS5 firewall, it requests a listening socket from the SOCKS5 firewall and specifies Client A's IP address as the binding address.

Client A then proceeds to connect to this address to chat. If Client A is behind a SOCKS firewall, it sends a connection request to it.

DCC Send Passive Protocol

Client A, initiating the DCC Send, sends the passive connection request below to Client B. The port is set to zero, and the id number is a unique integer identifying the connection:

```
   DCC SEND filename address port filesize id
```

If Client A is behind a firewall, the address is the ip address of its SOCKS firewall.

Client B receives the message and sets up a listening socket, and sends the IP address and port back to Client A, specifying the id number that identifies Client A's request:

```
   DCC SEND filename address port filesize id
```

If Client B is behind a SOCKS5 firewall, it requests a listening socket from the SOCKS5 firewall and specifies Client A's IP address as the binding address.

Client A then proceeds to connect to this address to begin the transfer. If Client A is behind a SOCKS firewall, it sends a connection request to it.

Note: The DCC Resume and Accept protocols in mIRC are also extended by adding the id number to the end of the CTCP message, but otherwise work in exactly the same way.

Other features

Agents
Voice Commands

Hotlinks

System Menu
Window Menu
Help Menu

Text Copy & Paste
Control Codes

File Server

Key Combinations
Command Line

Miscellaneous Stuff

Hotlinks

When you move your **mouse** in a window **over** text that is a website or email **address**, or a **nickname** of a certain type, the mouse pointer changes to a **finger** indicating that you can click, double-click, or right-click on that text to perform certain functions.

If you **double-click** on on a website address, mIRC will open up your web browser to visit that website.

You can also hold down the **shift-key** and **double-click** on email addresses to open your email program. The **shift-key** is required because of the huge number of addresses on IRC which look like emails but aren't.

If you're in a **channel** window and you move the mouse over text that is a **nickname** on that channel, you can **double-click** on it for the usual double-click behaviour, or **right-click** on it to open the nickname list popup menu. If you **single-click** on the nickname, the cursor in the listbox is scrolled to that nickname.

You can also **double-click** on text that is a **channel name** in any window to join that channel.

If you move the mouse over text that is a nickname in your **Notify** list, you can also click your **right** mouse button to pop up the notify popup menu.

Note: This behaviour depends on the Hotlink setting in the General dialog.

System menu

If you click the **system menu** button in the top left hand corner of a window ie. the button you usually double-click to close a window, it will popup the usual system menu but with a few added functions (these vary depending on the type of window).

Position
You can tell mIRC to **remember** or **forget** the position/size of a window. If you select remember, the next time that window opens up it will do so in the saved position. If you choose **reset**, the window will be moved back to it's previously saved position.

Note: If a window's saved position lies outside the size of the main window then it will open in a default position and size. To force a window to open up in default positions assigned by windows, select **forget**.

Buffer
You can **clear** the text in the current window buffer or you can **save** the text to a file.

Background
This allows you to select a background .BMP picture for the a window. You can choose to have the picture displayed in various ways, eg. tiled, centered, etc.

Note: Displaying a background picture slows down the display of text in a window significantly.

Spacing...
This allows you to set the line-spacing for messages in a window.

Save As...
This option is only available in custom picture windows, it allows you to save the current picture to a .bmp file.

Nicklist
This option is available in channel windows and allows you to change the position of the nicknames listbox.

Editbox
This option is available in channel windows and allows you to create a **second editbox**, for general use. You can press **Alt+Q** to show/hide the editbox. Also see a related option in the General dialog.

Font
This allows you to change the default font for a window. The font settings for each window will be remembered across sessions. Only the fonts which mIRC can display are listed in the fonts dialog.

Logging
If logging is turned on, any text displayed in a window will be logged to a file. This setting stays on across sessions until you switch it off. The filename is automatically taken from the name of the window.

Beeping
If beeping is turned on, mIRC will beep any time a message is sent to the window if it **isn't** active. This setting is remembered across sessions for each window.

Flashing
If flashing is turned on, the mIRC window/icon is flashed if there is a new message in the window while mIRC is not the active application. The flash sound specified in the Event Beeps section is also played.

Desktop
This allows you to position a window outside of the main mIRC window and onto the desktop.

Stay On Top
This appears only when a window is opened in Desktop mode and forces the window to stay above all other windows.

Timestamp
This turns timestamping of events on/off for a window.

Track URLs
This appears in Channel/Query windows, if turned on, mIRC auto-opens websites as they are mentioned in a message.

Events
This appears in Channel windows, it allows you to set display settings for events for a **specific** channel. The default display settings for **all** channels can be set via the IRC dialog.

Window Menu

The Tile, Cascade, and Arrange Icons menu items work the same way they usually do. If you want a **vertical** tiling of windows hold down the **Shift** key when you select tile.

The **Group by Network** option sorts the windows by network whenever you tile or cascade the windows.

The **Status to Top** option makes sure that the **status window** for the active connection is always **placed above** the status windows of other connections.

The **auto-cascade** and **auto-tile** options reposition/resize windows every time a new window is created or destroyed so as to make all of the windows more accessible.

The **auto-arrange** option re-arranges minimized icons (when the switchbar is turned off) whenever an icon window is closed.

A list of currently open windows is shown beneath these items.

Help Menu

The mIRC help menu is **dynamic** which means that mIRC looks in its **current** directory for files ending in **.hlp** and **.txt** and adds them to the **help** menu for easy access. It also makes **aliases** for each of the files listed in the help menu, the aliases being the **name** of the file **excluding** the extension.

For example, if you put a help file by the name of **winsock.hlp** in your mIRC directory, mIRC would add the **winsock.hlp** item to your help menu, and would make an alias **/winsock**. You can then type **/winsock sometext** and mIRC would do a **context-sensitive** search in that help file for the text you specified.

You can **add/remove** files whenever you want from the mIRC directory, mIRC always **updates** the help menu whenever you open it.

If you download the mIRC FAQ in the **.hlp** format into the mIRC directory, it will appear in this menu.

Text Copy & Paste

To copy text
You mark the text as usual with the mouse by pressing the left mouse-button and dragging it. The moment you release the left mouse-button, the text will be copied into the clipboard.

Note: See Key Combinations for keys you can use while copying text.

To paste text
You can then do the usual Shift-Insert key combination to paste the text anywhere you wish.

The limitation
You can only copy the currently displayed text. To copy text from another page, you must scroll up/down to it and then copy it. If you want to store most of the text you see on a channel, you might want to use the logfile/buffer options in the System Menu.

The explanation
The use of color in mIRC means that a simple text box cannot be used since text boxes can display only plain text (and they also have other limitations). However, text boxes also have built in cut/copy/paste routines which unfortunately are unavailable to a graphic window. This means that I had to code the mark/copy routine myself. I'm not sure which ran out first, my patience or my programming ability :-)

Control Codes

mIRC interprets control codes in text for **Bold, Underline, Reverse, and Color** and displays text in the specified format.

You can use the following key combinations to insert control codes in text:

Control+B for **bold** text
Control+U for **underlined** text
Control+R for **reverse** text
Control+K for **colored** text
Control+O for **plain** text

Examples

To **underline** a single word in a sentence:

1.Type Control+U
2.Type in the word
3.Type Control+U again

Only the text that is **enclosed** by the start and end codes will be affected. You can use this method with all of the other control codes.

The **Control+K** control code is slightly different because it allows you to specify a color number. To **color** a single word in a sentence:

1.Type Control+K
2.Type a number between 0 and 15
3.Type the word
4.Type Control+K again

If you also want to change the **background** color of a word, you would need to type **two numbers** separated by a **comma** instead of just one number. The first number is the **text** color, the second number is the **background** color. The colors range from 0 to 15, and the indexes are:

0 white	8 yellow
1 black	9 lightgreen
2 blue	10 cyan
3 green	11 lightcyan
4 lightred	12 lightblue
5 brown	13 pink
6 purple	14 grey
7 orange	15 lightgrey

If you want to enclose **existing** text in control codes, just **select** the text with your cursor, and then type the Control code. This will insert both starting and ending control codes around the text you selected.

You can enclose text in multiple control codes, so for example you could have a bold, underlined, and colored word.

You can use color 99 to indicate a transparent color.

Note: If you have the **Pop up color index** switch turned on in the Options dialog, mIRC will pop up a small color index showing you each color and it's associated number so you don't have to memorize them.

If you want to strip out control codes that other people send you in private or channel messages, you can either change the strip settings in the Messages dialog, or you can use the /strip command.

File Server

The mIRC fileserver allows other users to access files on your hard disk and is therefore **dangerous** since if used **improperly** it will allow them to access private/confidential information.

The /fserve command

A fileserver is initiated by using the **/fserve** command which initiates a DCC Chat to the specified user. You must specify a homedirectory. The user will be limited to accessing only files and directories within this homedirectory. The format is:

/fserve <nickname> <maxgets> <homedirectory> [welcomefile]

The **maxgets** is the maximum number of **simultaneous** dcc gets that a user can have during a fileserver session.

The **welcome file** is a text file that is sent to the user when they first connect. For example:

/fserve goat 5 c:\users\level1 level1.txt

This will initiate a filserver session to user goat with his homedirectory as c:\users\level1 and will send goat the text in the level1.txt file (presumably informing him that he is a level1 user and what files he can access etc.). The user can only have 5 simultaneous gets.

In each directory, you can place a **dirinfo.srv** file which describes that directory. Everytime the user does a CD to change into a directory, mIRC will look for this file and if it finds it, the text in it will be sent to the user.

Fileserver commands

The commands availabe to a user connected to your fileserver are:

cd <directory> - change to the specified directory.

dir [-b|k] [-#] [/w] - lists the name and size of each file in the current directory. The /w switch forces a wide listing. The [-b|k] selects bytes or k's. The [-#] specifies the number of files on each line in a horizontal listing.

ls [-b|k] [-#] - lists the name of each file in the currenty directory using a wide listing.

get <filename> - asks the fileserver to DCC Send the specified file.

read [-numlines] <filename.txt> - reads the specified text file. The user will be sent a default of 20 lines and then prompted whether to continue listing. The -numlines option changes the default number of lines to a value between 5 and 50.

help - lists the available commands.

exit or **bye** - terminates the connection.

Note:
1. If a directory has a large number of files try to split them up into subdirectories, this will improve performance.
2. If a user is idle for too long the fileserver will automatically close the connection. You can

set the idle time out in the DCC Options dialog.
3. A user is limited to opening a **single** fileserver session at any one time. If mIRC initiates a fileserver session to a user and that user doesnt respond then the fileserver session will have to time-out and close before that user can ask for another session.

Key Combinations

F1
Shows the help file and is context sensitive, so you can press it in a dialog and it will bring up the help file section describing that dialog. Remember that if this key is redefined as an alias it will no longer work as a help key.

Shift+F1
Displays the Keyword search dialog for the help file. If this is redefined as an alias it will no longer work as a help key.

Control+Tab
Switches between windows.

Shift+Tab
Switches between the editbox and the nickname listbox in a channel window.

Alt+Enter
In a multi-line editbox this moves the cursor to the next line allowing you to enter in several separate lines.

Control+F
In status/channel/query/etc. windows, this opens up a text search dialog allowing to you search the text buffer of that window for matching text.

Control+N
Cycles through channel windows.

Control+Q
Cycles through query windows.

Control+L
Scolls back text in a window to the Line Marker.

Tab
If an editbox is empty and you press the Tab key, mIRC inserts a "/msg nickname" into the editbox of the window you are currently on, where **nickname** is the last person that sent you a message.

You can use **Control+D** to remove a nickname from the Tab Key nickname list of users who have recently messaged you.

Note: The editbox needs to be empty in order for the Tab key to work as described above since the Tab key also performs the functions below...

If you are in a channel window editbox and it contains some text, the Tab key performs nickname completion on the word the cursor is on, this allows you type in only the first character or two of a nickname on that channel and then press Tab and mIRC will expand it to the full nickname.

If you press the Tab key while the cursor is positioned over a %variable or $identifier, it is evaluated.

The Tab completion also works for a nickname in a query window, the nicknames in your notify list when in the status window, and for channel names in your channels folder when in any window.

You can also use wildcard *? characters in nickname tab completion.

Cursor Up/Down
Browses the command line history buffer for a single-line editbox.

Control+Cursor Up/Down
Browses the command line history buffer for a multi-line editbox.

Page Up/Down
Browses the scrollback buffer of a window a page at a time.

Control+Page Up/Down
Browses the scrollback buffer of a window a line at a time.

ESCape
Quickly minimizes the active window, it must be turned on in the General dialog.

Shift+Copy
If you want to copy text with line-breaks as it appears in a window, you can hold down the **Shift** key while you do the copy.

Control+Copy
If you want to copy text with control codes, you can hold down the **Control** key while you do the copy.

Control+Enter
If you want to send information beginning with the **/** command prefix and you want it to be sent as normal text instead of **interpreted** as a command, just hold down the **Control** key when you press enter.

Control+B/U/R/K
Inserts control codes for bold, underline, reverse and color in text.

Alt+1...9
If you press Alt and a number, mIRC will display the Nth window listed in your Window menu. If you press Alt+0 (zero) and you are in an mIRC desktop window, it will jump to the main mIRC window.

Shift+Right-Click
You can roll/unroll a window by holding the shift-key and clicking your right mouse button on the window titlebar.

Control+Connect
You can hold down the Control key while clicking the Connect toolbar button to make mIRC use the next server in the list.

Shift+Connect
You can hold down the Shift key while clicking the Connect toolbar button to make mIRC connect to the current server using the same port instead of a randomly selected port.

Control+Minimize
This minimizes mIRC and asks you for a Lock password..

Shift+Minimize
This minimizes mIRC in a way opposite to that selected in the Tray dialog.

Alt+Q
Shows or hides the second editbox in a channel window.

Command Line Parameters

You can specify the following parameters on the command line:

-s<server:port> forces mIRC to connect to the specified server and port on startup.

-j<#chan1,...,#chanN> forces mIRC to join the specified channels on connect.

-p<password> specifies password required to join channel.

-n<nick1,nick2> sets your nickname and alternate nickname to these nicks.

-i<filename.ini> forces mIRC to use the specified INI file.

-r<rootdir> sets the root mIRC directory (by default this is where the mIRC EXE file is)

-service makes mIRC run as a service under 95/98.

-nouninst prevents mIRC from adding an uninstall option to the control panel add/remove dialog.

Miscellaneous stuff

Saving options
mIRC saves the main window position and all other settings when you exit the program. Note that the sizes (and not positions) of the finger, dcc get/chat/send, channel list, etc. windows are saved. The next time one of these windows opens up, it will be the same size as the last time you used it.

Toolbar
Some of the toolbar buttons have their own popup menus, you can click your right mouse button on the toolbar buttons to view them.

The Cookies
A squeaky sound, a multitude of silly comments, a faithful pet, one bouncing dot, and a smiley face. Where oh where can they be? :)

The Author (and part-time Human Bean)

Greetings! :)

I am **Khaled Mardam-Bey**, I can be reached via email at khaled@mirc.com.

One day, seven years ago, in a small, quiet room by myself, I stood in front of a canvas, picked up a brush, and began to work on a painting, dabbling here and there, not really knowing **what** I was going to create, or **how**.

A few years later, I put my brush down for a moment to pause for **breath**, look around, and find that I'm now in a **different** room, a much bigger one, filled with all kinds of people, with colors and sounds, and with **other** paintings, bigger and smaller than mine; some of the people are standing in front of me, close to the canvas, touching and **smudging** it, others are behind me looking at my painting from a distance; some are standing right **next** to me, telling me **which** colors to use or **how** to hold my brush, and criticising me whenever I make a mistake; a few are working on parts of the painting all by themselves.

mIRC today is still a **personal** work for me, and I think I've managed to keep working on it in the same **spirit** as when it all started; a mixture of fun, learning, and contribution. I've spent the last few years being part of a remarkable community made up of people from all over our planet, and that, in no small way, is part of what has kept me enjoying it, and believing in it.

It hasn't been easy reconciling mIRC as a personal work with its public use; I've had to learn to treat **some** aspects of it impersonally, which I'm not happy about, eg. I now use a pre-written reply to emails that ask the most common questions; if I tried to answer every email, I'd have little time for anything else.

I've also had to learn to accept **criticism**, which is hard for something that's personal; but mIRC wouldn't be where it is today without help from many people over the years.

These days I still enjoy working on mIRC, I'm getting a **bit** more tired now, but I think there may be a year or two left in me yet. I still get nervous before releasing a new version, which is probably a good sign ;) I hope you enjoy it.

I hope that mIRC has had, and will continue to have, a part to play in the making of new **friendships**, in the **keeping** of old ones, in the fostering of peaceful **communication**, and in the increased **understanding** and **respect** of other people and cultures, and that it has had a **positive** effect on people's lives.

I hope you enjoy using mIRC as much as I enjoyed creating it.

Khaled

License Agreement

mIRC is **shareware**, which means that you can use it freely for **30 days** to evaluate it. If during, or at the end of, the evaluation period you decide that you would like to continue using mIRC, please **register** your copy. Your single-user registration will license you to use your copy of mIRC, will support work on future versions, new features, and bug fixes, and will provide you with technical support via email.

Please see the How To Register section in the mIRC help file or on the mIRC website at http://www.mirc.com to find out how to register.

If you would like to **distribute** mIRC as part of a shareware distribution, magazine, book, cd-rom, website, etc., or you are interested in using mIRC in a **commercial** setting, eg. site license for a company, please email **khaled@mirc.com**.

mIRC may only be distributed in the original distribution install file as distributed by mIRC Co. Ltd. The mIRC distribution install file may not have files added to it or removed from it, and none of its contents may be modified, decompiled, or reverse engineered.

mIRC is provided **AS IS** without warranty of any kind, either express or implied, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. In no event shall mIRC Co. Ltd. be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages, even if mIRC Co. Ltd. has been advised of the possibility of such damages.

mIRC is a registered trademark of mIRC Co. Ltd.
The mIRC Logo is a trademark of mIRC Co. Ltd.

How to Register

As described in the License, mIRC is **shareware**, which means that you can use it freely for 30 days to **evaluate** it. If during, or at the end of, the evaluation period you decide that you would like to continue using mIRC, **please** register your copy. Your single-user registration will license you to use your copy of mIRC, will support work on future versions, new features, and bug fixes, and will provide you with technical support via email.

You can find the latest registration information in the **How To Register** section on the mIRC website at http://www.mirc.com

Acknowledgements - Thank you, thank you, and thank you...

After seven years of working on mIRC, it's hard to remember all of the people who've been involved with some aspect or other of its development; some I haven't heard from in a very long time, while others I still talk to today. They've all contributed in some way, whether through bug reports, suggestions, answering questions, beta-testing, helping out folks on IRC channels, or creating helpful scripts, documentation, and web pages.

I get emails about mIRC every day from people all over the world, emails which are often very touching and heartfelt, and I know that their kind words are as much a tribute to all of you who have helped to bring so many people together.

Thanks to...

Tjerk Vonck who created the first #mIRC channel, the mIRC homepage, and mIRC FAQ, and has contributed to mIRC in too many ways to mention.

The **mIRC Beta-testers**, past and present, for their hard work in catching bugs and suggesting features.

The **Helpers** on IRC channels who help out new users day after day, and create helpful webpages, documentation, and scripts.

Jarkko Oikarinen, creator of IRC, and all of the **ircd coders** after him who have worked hard to improve IRC for all of us.

Nicolas Pioch for his short IRC primer, **Ramji**, **Bibbly**, **Stimps**, **Mike**, **James G.**, **Edward**, **Bunster** for the queen-size futons, **Peeg** for the *bonk*s over the head, and **Viv** :)

Kevin Day, for coding and contributing various routines.

Andrzej Kowalik, for contributing his work on the toolbar icons.

M.Hunnibell, **M.Overtoom**, **Mark "Too Slick" Hanson**, **Ron R.**, and **Dan Lawrence** for their technical help.

Richard "Budman" Jones, who designed and contributed the logo.

Regular expression support is provided by the **PCRE** library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England.

JPEG support is based on the work of the Independent JPEG Group.

PNG support is based on the work of a variety of Authors and Group 42, Inc.

All of the other kind folks who have helped me out in many ways, and of course those who **supported** and **laughed** at my humble efforts :)